Master in Telecommunications Engineering

Master's Thesis

Line-Rate DDoS Detection System Using DPDK and OctoSketch

Author: Claudio Esteban Quesada

Director: Alan Zaoxing Liu

Madrid
April 2026

Declaration of originality

I declare under my responsibility that the Project presented with the title **Line-Rate DDoS Detection System Using DPDK and OctoSketch** at the ICAI School of Engineering of the Comillas Pontifical University in the academic year 2025/2026 of my authorship and has not been presented previously for other purposes. The Project is not plagiarised from any other, either totally or partially, and the information that has been taken from other documents is duly referenced

Use of Artificial Intelligence[1]

I declare under my responsibility that (indicate the correct option):

☐ I have not used Artificial Intelligence in the preparation of this document..

X I have used Artificial Intelligence in the preparation of this document and/or Annex B under the conditions allowed by Comillas Pontifical University, i.e. applying Level 2 of the Perkins et al. (2024) Assessment Scale: "AI can be used for pre-task activities such as brainstorming, description and initial research. This level focuses on the use of AI for planning, synthesising and generating ideas, but assessments should emphasise the ability to develop and refine these ideas independently". Specifically, Artificial Intelligence has been used to:

(indicar aquí el uso concreto que se ha hecho de la Inteligencia Artificial) Specifically, Artificial Intelligence has been usedito: • Research and Literature Review: To identify and summarize previous works, academic papers, and relevant research in the field. • Technological Brainstorming: To explore potential technologies for the project and evaluate their specific advantages and limitations

Signature (student): Claudio Esteban Quesada

[1] This declaration refers to the use of generative Artificial Intelligence to carry out the Project documents (Annex B and Memory). It does not apply to Projects where, by their nature, artificial intelligence must be used as part of them (application of machine learning techniques, neural networks, data analysis...)

Date:

Authorisation for Project delivery

Thesis supervisor	Thesis deputy supervisor (if any)
Zaoxing Liu	
Signature: Alan Zaoxing Liu	Signature: Claudio Esteban Quesada
Date: May 14, 2026	Date: May 14, 2026

Master in Telecommunications Engineering

Master's Thesis

Line-Rate DDoS Detection System Using DPDK and OctoSketch

Author: Claudio Esteban Quesada

Director: Alan Zaoxing Liu

Madrid
April 2026

Acknowledgements

To my family, for allowing me to pursue what I love. To my university friends, for teaching me so much along the way. To my ICAI colleagues at the University of Maryland, for six unforgettable months.

SISTEMA DE DETECCIÓN DE DDOS A TASA DE LÍNEA BASADO EN DPDK Y OCTOSKETCH

Autor: Claudio Esteban Quesada
Director: Alan Zaoxing Liu
Entidad Colabordora: University of Maryland

Resumen del Proyecto

En este Trabajo de Fin de Máster (TFM) se ha diseñado, implementado y validado sobre hardware de consumo un sistema de detección de ataques DDoS a velocidad de línea, empleando el Kit de Desarrollo del Plano de Datos (DPDK) y estructuras de datos probabilísticas OctoSketch. Se han desarrollado dos modos de detección complementarios: DPI-Sketch, que incorpora características derivadas de la carga útil de los paquetes para máxima precisión, y Sketch-ADV, que opera exclusivamente sobre cabeceras y permanece eficaz con independencia del cifrado del tráfico. Ambos modos emplean LightGBM para la clasificación multiclase de catorce categorías de tráfico. El sistema mantiene un procesamiento de hasta 16,90 Gbps sin pérdida de paquetes en hardware a través de 14 trabajadores paralelos, emitiendo las primeras alertas de detección entre 31 y 94 ms, más de un orden de magnitud más rápido que los sistemas comparables descritos en la literatura.

Palabras clave: detección de DDoS, DPDK, OctoSketch, estructuras probabilísticas, aprendizaje automático, procesamiento a velocidad de línea, seguridad de redes, tráfico cifrado.

1. Introducción

Los ataques de Denegación de Servicio Distribuida (DDoS) siguen siendo una de las amenazas más disruptivas para la infraestructura de red moderna, con volúmenes de ataque que superan regularmente varias decenas de gigabits por segundo. La detección de estos ataques en enlaces de alta velocidad plantea un reto fundamental: la pila de red del núcleo Linux procesa como máximo 3–5 Mpps en hardware de consumo, mientras que un enlace Ethernet de 25 Gbps con tramas de tamaño mínimo puede generar hasta 37 Mpps, es decir, un orden de magnitud por encima de la capacidad del núcleo.

Una restricción estructural adicional procede de la adopción generalizada de protocolos cifrados como TLS 1.3, QUIC y túneles VPN. Los enfoques que se basan en la Inspección Profunda de Paquetes (DPI) resultan inherentemente ineficaces cuando las cargas útiles son opacas. Un detector práctico debe abordar los cuatro retos de forma simultánea: operación a velocidad de línea, memoria acotada y fija, transparencia ante el cifrado, y precisión en la clasificación multiclase.

2. Definición del Proyecto

Este trabajo se propone diseñar e implementar un sistema de detección de DDoS que satisfaga seis objetivos específicos: (i) procesamiento de paquetes a velocidad de línea sin pérdidas en hardware; (ii) monitorización de tráfico con memoria fija y acotada mediante estructuras probabilísticas; (iii) ingeniería de características estructurada para representaciones tanto conscientes de la carga útil como exclusivas de cabecera; (iv) precisión de clasificación equivalente sin inspección de carga útil; (v) inferencia en tiempo real rápida, con detección en decenas de milisegundos; y (vi) validación reproducible en infraestructura de acceso público.

El sistema se construye sobre DPDK para E/S de paquetes sin paso por el núcleo, OctoSketch (variante Count-Min-Sketch de 8 filas × 4096 columnas, 640,1 KB por trabajador) para la agregación de tráfico por protocolo, y LightGBM para la clasificación embebida con gradient boosting. Todos los componentes son de código abierto y los experimentos se realizan en CloudLab utilizando tráfico sintético generado siguiendo la taxonomía de ataques CIC-DDoS2019.

3. Descripción del Sistema

El flujo de detección comprende tres capas funcionales, ilustradas en la Figura 3.

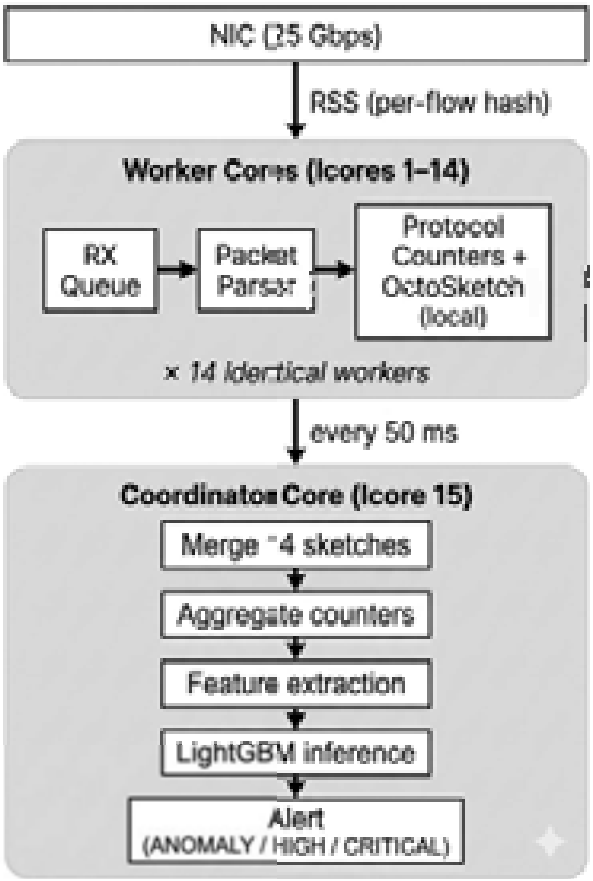

Figure 3: Arquitectura del sistema de detección DDoS con OctoSketch.

El **camino rápido** asigna los paquetes entrantes a uno de 14 hilos trabajadores DPDK mediante Receive Side Scaling (RSS). Cada trabajador actualiza su instancia local de OctoSketch, consumiendo solo ≈1,56% de los ciclos disponibles por paquete, y acumula contadores durante una ventana deslizante de 50 ms. **DPI-Sketch** construye un vector de características de 75 dimensiones combinando 47 contadores derivados de cabecera con 28 estadísticas de carga útil. **Sketch-ADV** emplea 12 instancias de sketch por protocolo para producir un vector de 64 dimensiones exclusivo de cabecera, estructuralmente inmune al cifrado. Al final de cada ventana, el **núcleo coordinador** alimenta el vector al modelo LightGBM y emite una alerta de nivel de confianza (NORMAL / ELEVADO / ALTO / CRÍTICO).

4. Resultados

La Tabla 2 resume el rendimiento de clasificación offline de ambos modos sobre el conjunto de datos recopilado en CloudLab.

Modo	Características	wF1 (%)	wF1 cifrado (%)	Δ (pp)
DPI-Sketch	75 (cabecera + carga útil)	99,81	87,35	−12,46
Sketch-ADV	64 (solo cabecera)	98,54	98,54	0,00

Table 2: Rendimiento de clasificación: DPI-Sketch vs. Sketch-ADV (LightGBM, taxonomía CIC-DDoS2019, 14 clases).

Ambos modos se desplegaron en el banco de pruebas de CloudLab frente a diversos escenarios de ataque en vivo (entre ellos, una inundación UDP a 16,90 Gbps, un Web-DDoS a 8,28 Gbps y un ataque multivector a 13,16 Gbps) sin ninguna pérdida de paquetes en hardware. DPI-Sketch emite su primera alerta entre 31 y 56 ms, gracias a la rápida acumulación de patrones de bytes de carga útil dentro de la primera ventana. Sketch-ADV alcanza confianza HIGH o CRITICAL entre 56 y 94 ms según el tipo de ataque, necesitando una o dos ventanas para que los ratios de sketch por protocolo converjan. Ambas cifras contrastan favorablemente con las latencias >800 ms descritas para sistemas comparables en la literatura.

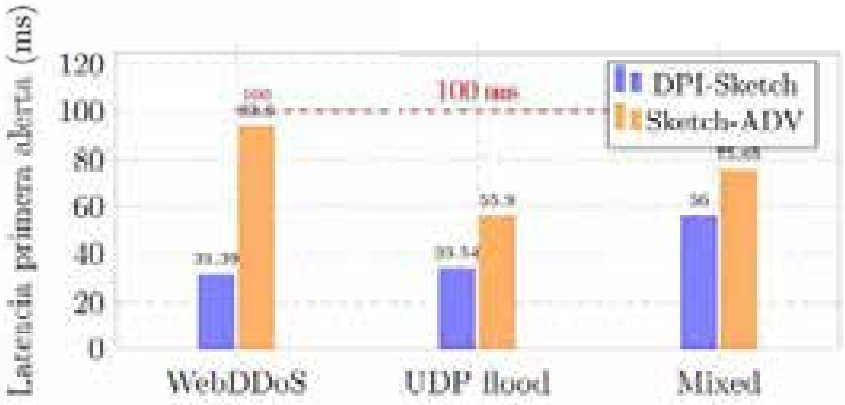

Figure 4: Latencia de primera alerta por tipo de ataque para DPI-Sketch y Sketch-ADV en el banco de pruebas en vivo.

5. Conclusiones

El sistema cumple los seis objetivos planteados. Mantiene operación a velocidad de línea hasta 16,90 Gbps sin pérdida de paquetes en hardware, conserva una huella de memoria acotada por trabajador (640,1 KB en DPI-Sketch, 8,1 MB en Sketch-ADV) con independencia de la intensidad del ataque, y emite la primera detección en menos de 100 ms en todos los escenarios en vivo. Sketch-ADV se identifica como la configuración de producción recomendada: 98,54% de F1 ponderado, inmunidad total al cifrado y cumplimiento de las restricciones legales y de privacidad que impiden la inspección de carga útil en un número creciente de entornos de producción. La brecha global de precisión respecto a DPI-Sketch se reduce a 1,27 pp y es operacionalmente despreciable para doce de las catorce clases de tráfico. El sistema completo está construido sobre componentes de código abierto y evaluado sobre un conjunto de datos público, garantizando su reproducibilidad sin necesidad de hardware ni software propietario.

6. Referencias

6.1. DPDK Contributors, *DPDK: Data Plane Development Kit*. https://www.dpdk.org, 2024.

6.2. G. Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Proc. NeurIPS*, 2017.

6.3. I. Sharafaldin et al., "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy," *Proc. IEEE ICISSP*, 2019.

6.4. R. Ricci et al., "Introducing CloudLab," *USENIX ;login:*, vol. 39, no. 6, 2014.

6.5. Y. Zhang, P. Chen, and Z. Liu, "OctoSketch: Enabling Real-Time, Continuous Network Monitoring over Multiple Cores," *Proc. 21st USENIX NSDI*, 2024, pp. 1621–1639.

LINE-RATE DDOS DETECTION SYSTEM USING DPDK AND OCTOSKETCH

Author: Claudio Esteban Quesada
Director: Alan Zaoxing Liu

Entidad Colaboradora: University of Maryland

Abstract

In this Master's Thesis (TFM), a line-rate DDoS detection system has been designed, implemented, and validated on commodity hardware using the Data Plane Development Kit (DPDK) and OctoSketch probabilistic data structures. Two complementary detection modes have been developed: DPI-Sketch, which incorporates payload-derived features for maximum accuracy, and Sketch-ADV, which operates exclusively on packet headers and remains effective regardless of traffic encryption. Both modes employ LightGBM for multi-class classification of fourteen traffic categories. The system sustains processing at up to 16.90 Gbps with zero hardware packet loss across 14 parallel workers, delivering first detection alerts within 31–94 ms, more than an order of magnitude faster than comparable systems reported in the literature.

Keywords: DDoS detection, DPDK, OctoSketch, probabilistic sketches, machine learning, line-rate processing, network security, encrypted traffic.

1. Introduction

Distributed Denial of Service (DDoS) attacks remain among the most disruptive threats to modern network infrastructure, with peak attack volumes regularly exceeding tens of gigabits per second. Detecting these attacks at high-speed links poses a fundamental engineering challenge: the conventional Linux kernel networking stack processes at most 3–5 Mpps on commodity hardware, while a 25 Gbps Ethernet link carrying minimum-size frames can generate up to 37 Mpps, an order of magnitude beyond the kernel's processing capacity.

A further structural constraint arises from the widespread adoption of encrypted protocols such as TLS 1.3, QUIC, and VPN tunnels. Detection approaches that rely on Deep Packet Inspection (DPI) are inherently ineffective once payloads are opaque. A practical detector must therefore address all four challenges simultaneously: line-rate operation, bounded fixed memory, encryption transparency, and multi-class classification accuracy.

2. Project's Definition

This work sets out to design and implement a DDoS detection system satisfying six specific objectives: (i) sustained line-rate packet processing without hardware drops;

(ii) bounded, fixed-memory traffic monitoring using probabilistic sketches; (iii) structured feature engineering for both payload-aware and header-only representations; (iv) near-equivalent classification accuracy without payload inspection; (v) fast inline real-time inference, with detection in tens of milliseconds; and (vi) reproducible testbed validation on publicly accessible infrastructure.

The system is built on DPDK for kernel-bypass packet I/O, OctoSketch (an 8-row × 4096 column Count-Min Sketch variant, ≈40.1 KB per worker) for per-protocol traffic aggregation, and LightGBM for embedded gradient-boosted classification. All components are open-source, and experiments are conducted on CloudLab using synthetically generated traffic modelled after the CIC-DDoS2019 attack taxonomy.

3. System Description

The detection pipeline comprises three functional layers, illustrated in Figure 1.

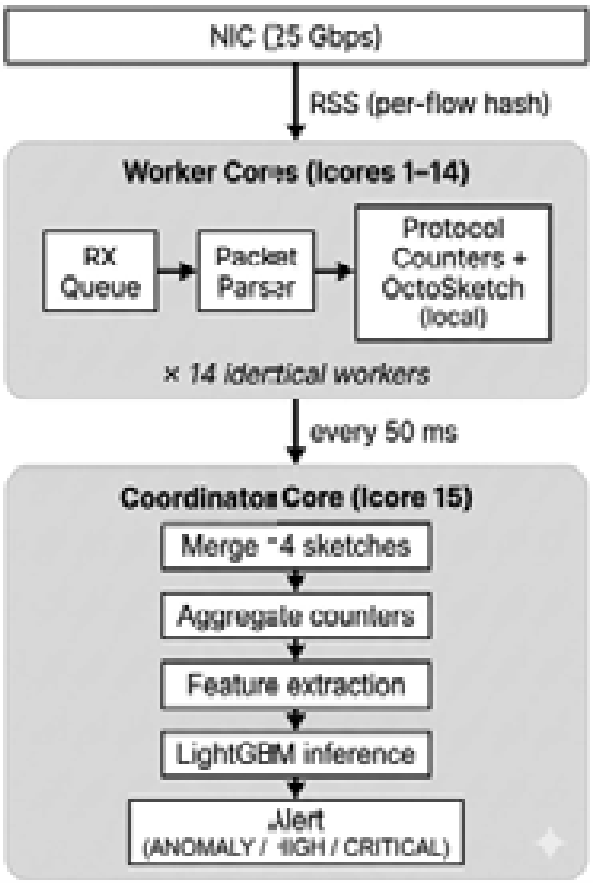

Figure 1: End-to-end architecture of the OctoSketch DDoS detection system.

The **fast path** assigns incoming packets to one of 14 parallel DPDK worker threads via Receive Side Scaling (RSS). Each worker updates its local OctoSketch instance, consuming only ≈1.56% of fast-path cycles per packet, and accumulates counters over a 50 ms sliding window. **DPI-Sketch** constructs a 75-dimensional feature vector

combining 47 header-derived counters with 28 payload statistics. **Sketch-ADV** uses 12 per-protocol sketch instances to produce a 64-dimensional header-only vector that is structurally unaffected by encryption. At the end of each window the **coordinator core** feeds the vector to LightGBM and raises a NORMAL / ELEVATED / HIGH / CRITICAL alert.

4. Results

Table 1 summarises the offline classification performance of both modes on the CloudLab-collected dataset.

Mode	Features	wF1 (%)	wF1 encrypted (%)	Δ (pp)
DPI-Sketch	75 (header + payload)	99.81	87.35	−12.46
Sketch-ADV	64 (header only)	98.54	98.54	0.00

Table 1: Classification performance: DPI-Sketch vs. Sketch-ADV (LightGBM, CIC-DDoS2019-based taxonomy, 14-class).

Both modes were deployed on the CloudLab testbed against various live attack scenarios (including a UDP flood at 16.90 Gbps, a WebDDoS at 8.28 Gbps, and a multi-vector mixed attack at 13.16 Gbps) with zero `imissed` NIC drops throughout. Figure 2 compares the first-alert latency of both modes across the three attack types. DPI-Sketch raises its first alert at 31–56 ms, driven by rapid accumulation of payload byte patterns within the first window. Sketch-ADV reaches HIGH or CRITICAL confidence at 56–94 ms depending on attack type, requiring one to two windows for per-protocol sketch ratios to converge. Both figures compare favourably against the >800 ms latencies reported for comparable systems in the literature.

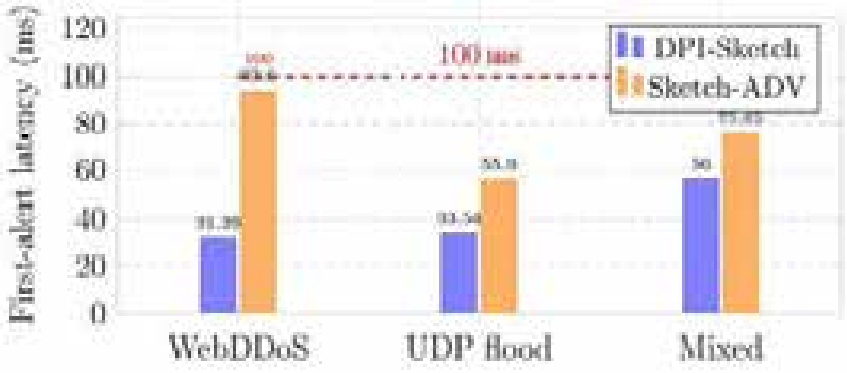

Figure 2: First-alert latency per attack type for DPI-Sketch and Sketch-ADV on the live testbed.

5. Conclusions

The system meets all six stated objectives. It sustains line-rate operation at up to 16.90 Gbps with zero hardware packet loss, maintains a bounded per-worker memory footprint (640.1 KB for DPI-Sketch, 8.1 MB for Sketch-ADV) regardless of attack intensity, and delivers first detection within 100 ms in all live scenarios. Sketch-ADV is identified as the recommended production configuration: 98.54% weighted F1, full encryption immunity, and compliance with privacy constraints that preclude payload inspection in a growing share of production environments. The 1.27 pp global accuracy gap relative to DPI-Sketch is operationally negligible for twelve of the fourteen traffic classes. The entire system is built on open-source components and evaluated on a publicly available dataset, ensuring full reproducibility without proprietary appliances.

6. References

6.1. DPDK Contributors, *DPDK: Data Plane Development Kit*. https://www.dpdk.org, 2024.

6.2. G. Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Proc. NeurIPS*, 2017.

6.3. I. Sharafaldin et al., "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy," *Proc. IEEE ICISSP*, 2019.

6.4. R. Ricci et al., "Introducing CloudLab," *USENIX ;login:*, vol. 39, no. 6, 2014.

6.5. Y. Zhang, P. Chen, and Z. Liu, "OctoSketch: Enabling Real-Time, Continuous Network Monitoring over Multiple Cores," *Proc. 21st USENIX NSDI*, 2024, pp. 1621–1639.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Distributed Denial of Service (DDoS) attacks remain one of the most prevalent and damaging threats to modern network infrastructure. These attacks aim to exhaust the resources of a target system, such as bandwidth, processing capacity, or memory, by flooding it with massive volumes of malicious traffic from multiple distributed sources. According to recent industry reports [1, 2], DDoS attacks have grown both in frequency and in magnitude, with peak attack volumes regularly exceeding tens of gigabits per second and, in extreme cases, reaching the terabit scale.

The detection of DDoS attacks at high-speed network links poses a fundamental engineering challenge. The conventional Linux kernel networking stack processes packets through a series of interrupts, context switches, and memory copies that limit its throughput to approximately 3 to 5 million packets per second (Mpps) on commodity hardware [3]. However, a 25 Gbps Ethernet link carrying minimum-size packets (64 bytes) can generate up to 37 Mpps [4], which is an order of magnitude beyond the kernel's processing capacity. This gap between the required and achievable packet processing rates means that a kernel-based detector would inevitably drop a significant portion of the traffic, leaving attacks undetected.

Existing approaches to DDoS detection can be broadly classified into two categories. On one hand, threshold-based systems apply simple rules over traffic statistics (e.g., packets per second exceeding a predefined value) and can operate at high speed, but they lack the ability to accurately classify different attack types and are prone to false positives [5]. On the other hand, machine learning-based systems can distinguish among multiple attack categories with high accuracy, but they typically rely on offline or near-real-time processing with detection latencies exceeding 800 ms, as reported in

recent works such as MULTI-LF [6]. Such latencies are unacceptable in environments where even a few hundred milliseconds of undetected attack traffic can cause significant service degradation.

A third challenge concerns the memory and computational cost of tracking per-flow traffic state. A high-speed link may carry millions of concurrent flows; storing exact counters for each flow in a conventional hash table requires memory proportional to the number of active flows. At a rate of 14 Mpps on a 10 Gbps link, a table with 4 M entries and 64 B of state per entry already occupies 256 MB, and the random access patterns of hash lookups cause frequent last-level cache misses that degrade throughput as packet rates increase. A practical detection system must therefore summarise traffic statistics in a data structure whose memory footprint is fixed and small, whose access pattern is cache-friendly, and whose update cost is $O(1)$ per packet regardless of the number of active flows.

A further structural constraint comes from the widespread adoption of encrypted protocols such as TLS 1.3 [7], QUIC [8], and VPN tunnels, which now account for the majority of Internet traffic [9]. Detection approaches that rely on Deep Packet Inspection (DPI), that is, the examination of packet payload contents, are inherently incompatible with encryption: once the payload is opaque, payload-derived features become unavailable regardless of computational resources. This structural incompatibility motivates a detection architecture that derives all traffic features exclusively from packet headers, such as source IP address, destination port, protocol field, and packet size, which remain visible even in fully encrypted sessions.

These challenges define the problem addressed in this work: the design and implementation of a DDoS detection system capable of operating at line rate on high-speed network links, using embedded machine learning classification to accurately identify multiple attack types, with a dedicated header-only operating mode that ensures the system remains applicable independently of payload encryption. The system must process all incoming packets inline, without sampling or mirroring, and deliver detection decisions within tens of milliseconds, not seconds.

Chapter 2

Technology Background

This chapter provides a detailed description of the technologies, protocols, tools, and libraries specifically used throughout the project, with the aim of facilitating its understanding and follow-up.

2.1 DPDK, Data Plane Development Kit

The Data Plane Development Kit (DPDK) is an open-source framework, originally developed by Intel and subsequently donated to the Linux Foundation, that provides a set of libraries and Poll-Mode Drivers (PMDs) for high-speed packet processing entirely in user space [3]. Its central goal is to bypass the Linux kernel networking stack, eliminating the overhead imposed by interrupt handling, context switches, and memory copies, so that a standard multi-core server can sustain packet processing rates in the tens of millions of packets per second on commodity hardware.

2.1.1 The Linux Kernel Networking Bottleneck

In the conventional packet processing model, the arrival of a packet at the NIC triggers a hardware interrupt that preempts the currently executing process and transfers control to a kernel interrupt handler. The handler copies the packet from the NIC's DMA region into a kernel socket buffer (`sk_buff`), which is later transferred to user space via a system call. Each step in this chain carries a non-trivial cost: a hardware interrupt forces a context switch that flushes TLB entries and evicts the L1/L2 cache lines warm for the interrupted process, requiring hundreds of nanoseconds of cache rewarming before useful work resumes. Across the millions of packets per second arriving on a high-speed link, these per-interrupt penalties accumulate into a hard throughput ceiling of approximately 1 Mpps to 3 Mpps on commodity hardware [3].

The Linux kernel introduced *NAPI* (New API) as a partial mitigation: after the first hardware interrupt signals packet arrival, subsequent packets are harvested in a software polling loop until the queue drains, amortising the interrupt cost over a batch of packets rather than paying it per packet [10]. While NAPI substantially improves throughput at moderate rates, it does not eliminate the `sk_buff` allocation, the kernel-to-user memory copy, or the system call overhead — all of which remain on the per-packet critical path for any user-space application. A 25 Gbps Ethernet link carrying minimum-size 64-byte frames demands up to 37 Mpps [4], a rate at which even NAPI's batching provides insufficient relief and the kernel networking stack becomes the dominant bottleneck.

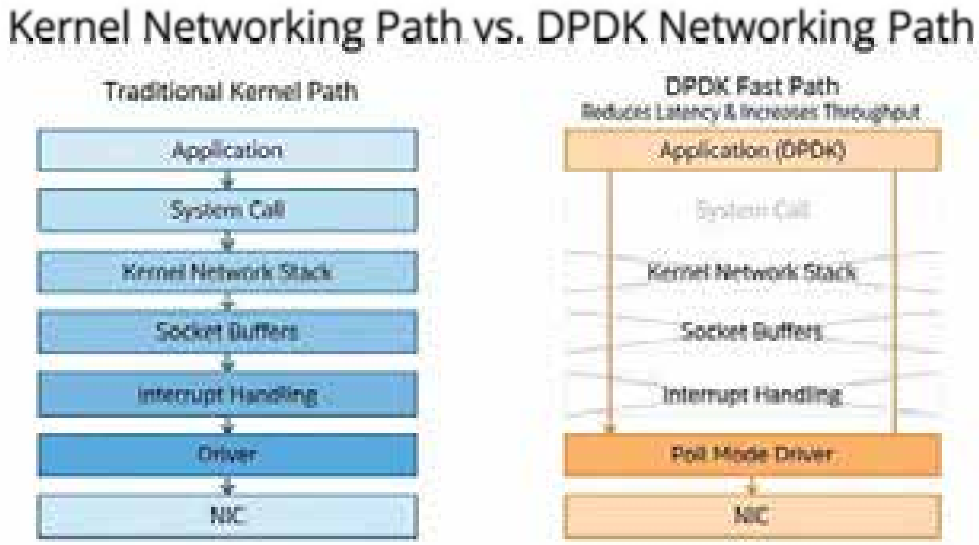

Figure 2.1: Kernel packet path vs. DPDK userspace path.

2.1.2 Environment Abstraction Layer

The core of DPDK is the Environment Abstraction Layer (EAL), which runs at application startup to perform low-level initialisation and resource reservation. Its responsibilities span four areas [3]:

- **CPU core affinity.** The EAL binds each logical worker thread (*lcore*) to a dedicated CPU core using `pthread` affinity settings, preventing the OS scheduler from migrating threads and eliminating the cache-rewarming overhead of cross-core migrations. The lcore-to-CPU mapping is specified at launch via the `-l` argument (e.g., `-l 0-3` to use cores 0 through 3) or the more flexible `--lcores` option that allows explicit core pinning.

- **Hugepage reservation.** The EAL allocates large memory pages of 2 MB or 1 GB from the operating system rather than the default 4 KB pages. Hugepages

reduce the number of TLB entries required to map packet buffer memory, lowering TLB miss rates at high packet rates, and they provide physically contiguous memory regions suitable for NIC DMA transfers.

- **PCI device detection and PMD binding.** On startup the EAL scans the PCI bus to enumerate network interfaces, then binds each NIC to its corresponding Poll-Mode Driver using the kernel's UIO or VFIO subsystem. This binding transfers ownership of the NIC from the kernel driver to the DPDK PMD, making the device invisible to the OS but fully accessible to the DPDK application.

- **Service initialisation.** The EAL initialises the logging subsystem, the high-resolution timer service (`rte_timer`), and the inter-process communication channel used to share resources between DPDK processes running on the same host.

2.1.3 Poll-Mode Drivers and Zero-Copy Packet I/O

Instead of waiting for a hardware interrupt to signal packet arrival, DPDK Poll-Mode Drivers keep one CPU core in a tight loop continuously polling the NIC's receive descriptor rings. The receive path works as follows: when packets arrive, the NIC writes their physical addresses into the RX descriptor ring via DMA; the PMD reads those descriptors in batches using `rte_eth_rx_burst()`, returning up to N pointers to `rte_mbuf` structures already populated with packet data, with no copy ever made. Processing the burst amortises the overhead of pointer dereferences, branch mispredictions, and cache-line loads across all N packets, which is why burst sizes of 16–32 are empirically optimal for most workloads.

Packets are managed through a *mempool*, a pre-allocated, lock-free pool of fixed-size mbufs backed by hugepage memory. Once the application has finished processing a packet, it returns the mbuf to the pool for reuse, avoiding dynamic memory allocation entirely.

The transmit path is symmetric: to send packets, the application fills an array of mbuf pointers and calls `rte_eth_tx_burst()`, which writes the corresponding physical addresses into the NIC's TX descriptor ring and signals the NIC to initiate DMA. The PMD then reclaims completed TX descriptors and frees the mbufs back to the mempool once the NIC confirms transmission. This *zero-copy* design on both RX and TX paths, combined with the removal of interrupt latency, is the primary driver of DPDK's throughput advantage over the kernel stack [3].

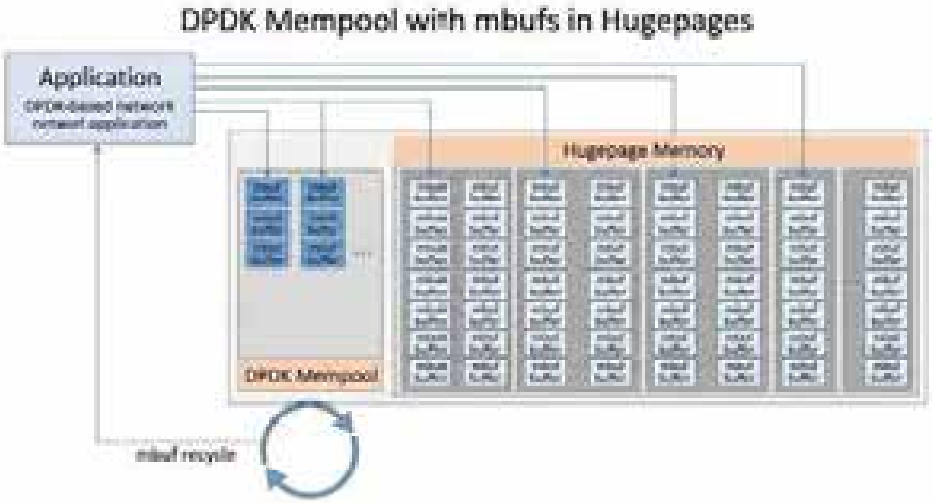

Figure 2.2: DPDK memory model: hugepage-backed mempool and zero-copy packet I/O.

2.1.4 Multi-Core Scaling and Receive-Side Scaling

DPDK applications typically follow one of two execution models for distributing work across cores:

- **Run-to-completion.** Each packet is received, processed, and optionally transmitted entirely within a single core, with no handoff to another thread. This eliminates inter-core synchronisation on the critical path and is the dominant model for inspection and forwarding workloads where per-packet processing is bounded in time.

- **Pipeline.** The processing of a packet is split across multiple stages, each executed by a different core. Stage outputs are passed between cores via lock-free ring buffers (`rte_ring`). The pipeline model is advantageous when different processing stages have heterogeneous computation costs or when one stage (e.g., cryptographic processing) benefits from being isolated on dedicated cores.

To spread incoming traffic across multiple cores, modern NICs implement *Receive-Side Scaling* (RSS), a hardware feature that computes a Toeplitz hash over packet header fields and maps the result to one of several independent hardware receive queues [3]. Each queue is assigned to a dedicated DPDK worker thread on its own core. The set of header fields that feeds the Toeplitz hash, the *RSS hash key*, is configurable: typical deployments hash on the full five-tuple (source/destination IP, source/destination port, protocol) to ensure that packets from the same flow always reach the same core, preserving processing order. For workloads that track per-source-IP statistics rather than per-flow state, the key can be restricted to the source IP address alone, producing

a coarser but sufficient distribution.

Because packets belonging to the same configured key always hash to the same queue, per-flow state can be maintained in core-local memory without any locking. DPDK also enforces *NUMA awareness*: memory pools are allocated on the same physical processor socket as the core that will access them, avoiding expensive cross-socket DRAM accesses that would otherwise become a throughput bottleneck on multi-socket servers.

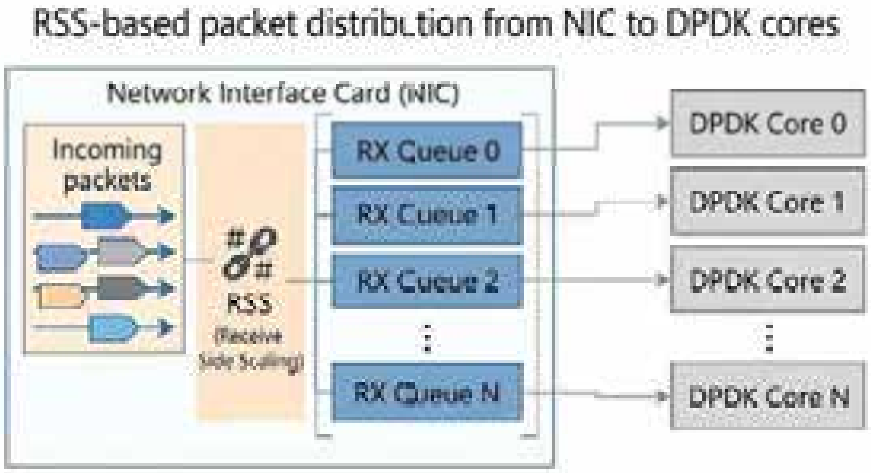

Figure 2.3: DPDK multi-core architecture with Receive-Side Scaling (RSS).

2.1.5 Performance and Applicability

Under the conditions described above, a commodity server equipped with a modern multi-core processor can sustain forwarding and inspection rates that scale approximately linearly with the number of dedicated cores [3]. The achievable throughput is strongly influenced by packet size: smaller packets require more packets per second to fill a given bit-rate link, increasing the per-core packet processing budget. Table 2.1 shows the line-rate packet demands at common frame sizes for a 25 Gbps link together with representative single-core DPDK forwarding throughput.

Table 2.1: Line-rate packet demands and representative single-core DPDK throughput at 25 Gbps.

Frame size (bytes)	Line-rate (Mpps)	Single-core DPDK forwarding
64	~37	2–3 cores for full line rate
128	~21	1–2 cores sufficient
256	~11	1 core sufficient
512	~6	Well within single-core budget
1500	~2	Well within single-core budget

DPDK has consequently become the standard foundation for network function virtualisation (NFV), software-defined networking (SDN) data planes, and inline traffic analysis. Notable production deployments include Cisco's Vector Packet Processing (VPP) [11], a high-performance software router used in telecommunications NFV deployments; Open vSwitch with DPDK acceleration, widely used in cloud hypervisor environments; and NVIDIA DOCA, the application framework for BlueField DPUs that exposes DPDK as its primary packet processing API. These adoptions confirm that DPDK's performance model is robust across diverse real-world workloads beyond the benchmarking context in which it was developed.

2.2 Probabilistic Data Structures: Count-Min Sketch and OctoSketch

High-speed packet processing leaves very little time per packet for complex data structure operations. This section describes the family of probabilistic data structures known as *sketches*, which summarise traffic statistics in a fixed, small memory footprint with guaranteed error bounds. Two structures are discussed: the Count-Min Sketch, which provides the theoretical foundation, and OctoSketch, the specific implementation adopted in this work.

2.2.1 Sketches and Stream Summaries

A *sketch* is a compact probabilistic summary of a data stream. Instead of storing the exact value associated with each key, a sketch maps keys to a small array of counters using one or more hash functions, allowing multiple keys to share the same counter. The result is a data structure whose memory footprint is fixed and independent of the number of distinct keys in the stream, at the cost of a small, controllable estimation error caused by hash collisions.

Sketch-based structures offer three properties that make them particularly well suited to high-speed network traffic monitoring. First, both update and query operations run in $O(1)$ time regardless of the number of active flows, making per-packet processing cost predictable and bounded. Second, the memory requirement is set at initialisation time by choosing the array dimensions, so no dynamic allocation is needed during packet processing. Third, sketches are *mergeable*: the sketch maintained by one CPU core can be combined with the sketch of another core by element-wise addition, enabling a natural parallelism model where each worker thread maintains its own local sketch and a coordinator thread periodically merges them without any locking.

The key trade-off is between memory and accuracy. A larger array reduces the prob-

ability of hash collisions and therefore reduces estimation error, while a smaller array saves memory. In practice, sketch dimensions are chosen so that the error is bounded well below the detection thresholds used in the application.

2.2.2 The Count-Min Sketch

The Count-Min Sketch, proposed by Cormode and Muthukrishnan [12], is the foundational structure of this family. It consists of a two-dimensional array of integer counters with d rows and w columns, and d independent hash functions $h_1, h_2, \ldots, h_d$, each mapping a key to one of the w column positions. The structure defines the following set of operations and guarantees:

- **Initialisation.** All $d \times w$ counters are set to zero. The d hash functions are chosen independently and uniformly at random from a pairwise-independent family, one per row. A common choice for high-speed implementations is the Toeplitz hash, the same hash family used by NIC hardware for RSS (Section 2.1), which can be computed efficiently with bitwise XOR and shift operations, requiring no multiplication or division on the critical path.

- **Update.** When a new element with key x arrives, for each row $i \in \{1, \ldots, d\}$ the counter at the column indicated by $h_i(x)$ is incremented by one (or by a weight v for weighted streams such as byte counts):

$$C[i,\, h_i(x)] \mathrel{+}= v \quad \forall i \in \{1, \ldots, d\}$$

- **Query.** To estimate the frequency $\hat{f}(x)$ of key x, the sketch returns the minimum counter value across all rows:

$$\hat{f}(x) = \min_{1 \le i \le d} C[i,\, h_i(x)]$$

  The minimum is taken because hash collisions can only *increase* a counter, never decrease it. The true frequency is therefore always at most the minimum observed value, making the estimate a guaranteed overestimate.

- **Merge.** Two sketches of identical dimensions built over disjoint streams can be merged into a single sketch representing the union of both streams by element-wise addition of their counter matrices. This property enables embarrassingly parallel deployments where each thread maintains a local sketch and a central thread periodically aggregates them.

- **Error guarantees.** With probability at least $1 - \delta$, the point query satisfies

$\hat{f}(x) \leq f(x) + \varepsilon \|f\|_1$, where $\|f\|_1$ is the total weight inserted. Setting $w = \lceil e/\varepsilon \rceil$ and $d = \lceil \ln(1/\delta) \rceil$ achieves these bounds. For example, targeting $\varepsilon = 0.01$ (1% of the total stream weight) and $\delta = 0.01$ yields $w = \lceil e/0.01 \rceil \approx 272$ columns and $d = \lceil \ln 100 \rceil = 5$ rows — a structure of roughly 5 KB that fits comfortably in L1 cache. In practice, wider arrays of $d = 4$ to 8 rows and $w = 1024$ to 8192 columns are common, trading a modest amount of memory for tighter error bounds.

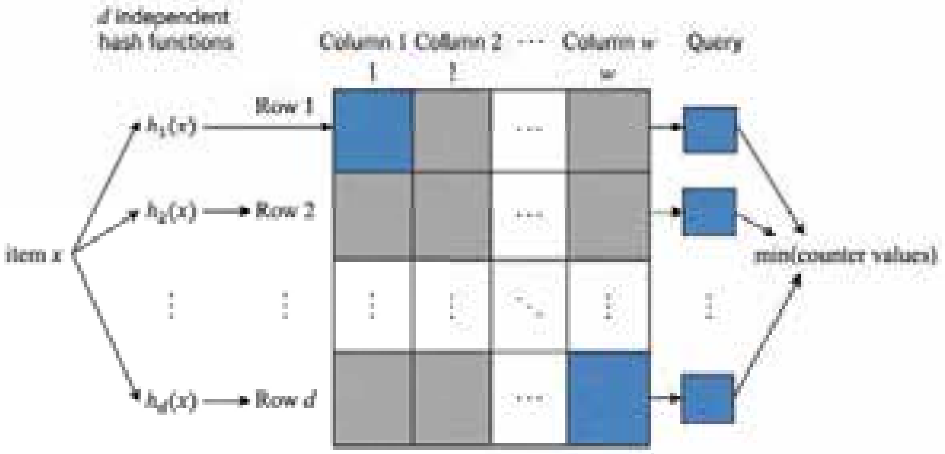

Figure 2.4: Count-Min Sketch structure.

2.2.3 OctoSketch

OctoSketch [13] is a software monitoring framework proposed by Zhang, Chen, and Liu (NSDI 2024) that addresses a fundamental limitation of sketch-based multi-core monitoring: the loss of online accuracy caused by periodic full merges. In a conventional multi-core sketch deployment, each core maintains a local sketch and a coordinator periodically collects and sums them to obtain a global view. The aggregated result is only accurate at the moment of the merge; any query issued between two merge operations reflects a stale or incomplete snapshot of the traffic, as illustrated in Figure 2.5.

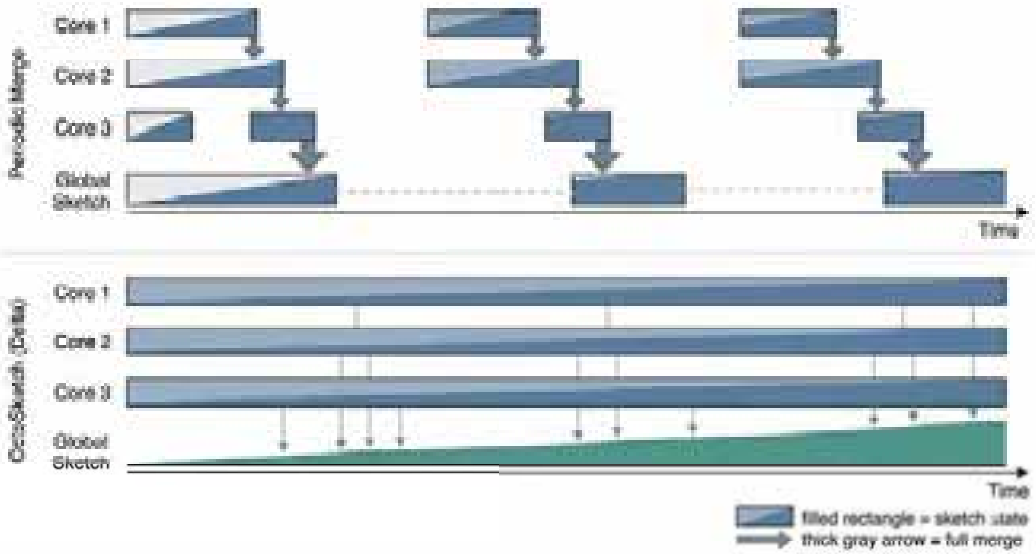

Figure 2.5: Periodic full-merge (top) vs. OctoSketch continuous delta aggregation (bottom).

OctoSketch solves this with a *continuous, delta-based aggregation* mechanism that operates as follows:

- **Shadow copy.** Each worker core maintains two sketch instances: the *active* sketch, which is updated on every incoming packet, and a *shadow* copy that mirrors the state last propagated to the coordinator.
- **Delta computation.** At each propagation step, the core computes the cell-wise difference between the active sketch and the shadow: $\Delta C[i,j] = C_{\text{active}}[i,j] - C_{\text{shadow}}[i,j]$. Only non-zero deltas need to be transmitted, keeping the communication volume proportional to the number of cells that changed since the last synchronisation rather than the full sketch size.
- **Global accumulation.** The coordinator adds each received delta directly to the global sketch: $C_{\text{global}}[i,j] \mathrel{+}= \Delta C[i,j]$. Because the global sketch is updated incrementally, it reflects the cumulative contribution of all cores at any point in time, without waiting for a full collection cycle.
- **Shadow update.** After propagation, the shadow is set equal to the active sketch, so the next delta captures only the changes made since the current synchronisation.

A key property of OctoSketch is its *generality*: it functions as a framework that wraps existing sketch algorithms, including Count-Min Sketch, Count Sketch, and others, without modifying their internal structure. The name *Octo* refers to the use of 8 hash rows in the underlying sketch, a configuration chosen by the authors to balance

accuracy and cache efficiency across a wide range of workloads. The paper evaluates OctoSketch against nine representative sketches on three platforms (CPU, DPDK, and eBPF XDP), reporting up to 4.5 × higher throughput and 15.6 × lower estimation error compared to periodic-merge baselines on DPDK deployments [13].

2.3 Machine Learning for Network Traffic Classification

Machine learning (ML) has become the standard approach for network traffic classification tasks where the number of traffic categories and the variability of attack patterns make hand-crafted rule systems impractical to maintain. This section describes the foundational concepts of supervised classification and the family of tree-based ensemble methods most relevant to this domain.

2.3.1 Supervised Classification

Supervised learning is the branch of machine learning concerned with learning a mapping $f : \mathcal{X} \to \mathcal{Y}$ from a set of labelled training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where $\mathbf{x}_i \in \mathcal{X}$ is a feature vector and $y_i \in \mathcal{Y}$ is the corresponding class label. In a classification setting, $\mathcal{Y}$ is a finite set of discrete categories, such as *benign*, *DNS amplification*, or *SYN flood*. The goal of training is to find the parameters of f that minimise a loss function measuring the discrepancy between predicted and true labels on the training set, while retaining the ability to generalise to unseen examples.

Once trained, the model is used in an *inference* phase: given a new, unlabelled feature vector $\mathbf{x}$, it produces a predicted class $\hat{y} = f(\mathbf{x})$. The quality of classification is assessed using metrics derived from the confusion matrix:

- **Accuracy:** the fraction of correctly classified instances. This metric is misleading when class distributions are skewed, a classifier that always predicts the majority class can achieve high accuracy while detecting nothing.
- **Precision and recall:** precision measures the fraction of positive predictions that are correct; recall measures the fraction of actual positives that are detected. The two metrics are in tension: raising the detection threshold increases precision but reduces recall.
- **F1 score:** the harmonic mean of precision and recall, $F_1 = 2 \cdot \frac{P \cdot R}{P + R}$. For multi-class problems with imbalanced distributions — such as network datasets where benign traffic dominates, the macro-averaged or weighted-averaged F1 across all classes is the standard summary metric [14].

To assess generalisation, the labelled dataset is partitioned into a *training set* used to fit the model and a held-out *test set* used exclusively for final evaluation. Hyperparameters are tuned on a separate *validation set* (or via cross-validation folds drawn from the training data) to avoid contaminating the test set with model selection decisions.

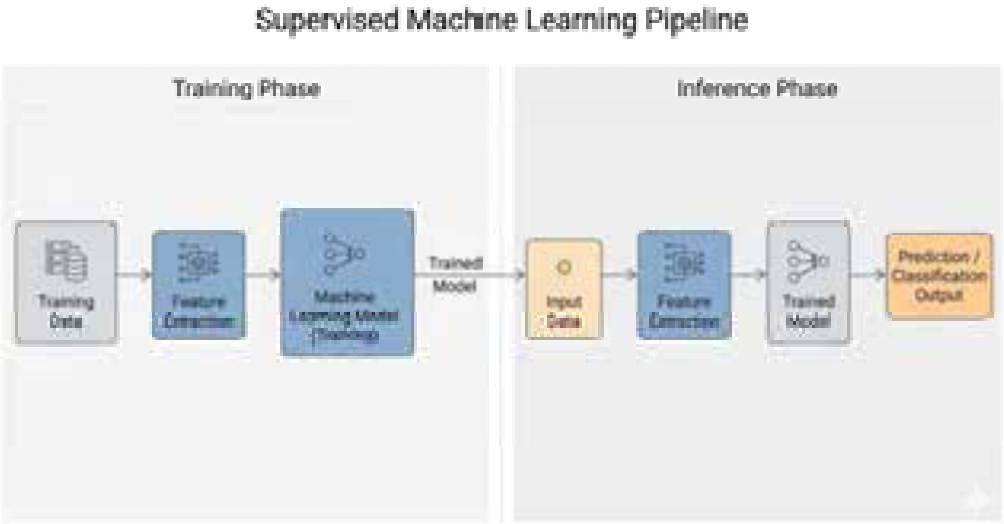

Figure 2.6: Supervised learning pipeline.

2.3.2 Feature Representation of Network Traffic

A feature vector is a fixed-length numerical representation of one observation. For network traffic classification, observations are typically defined in one of two ways:

- **Time-window granularity:** all packets arriving within a fixed interval (e.g., 50–100 ms) are aggregated into a single feature vector, regardless of flow boundaries. This approach captures aggregate traffic intensity and is well suited to volumetric attack detection, as it does not require maintaining per-flow state that may grow unboundedly under a flooding attack.

- **Flow granularity:** packets sharing the same five-tuple (source/destination IP, source/destination port, protocol) are grouped into a flow and tracked until a timeout or connection teardown. Flow-based features are richer and capture per-connection behaviour, but require scalable flow-tracking infrastructure and introduce latency proportional to flow duration.

Regardless of granularity, features summarise the traffic observed during the measurement period. Common feature categories include:

- **Volume features:** total packet count, byte count, and packets per second within the window.

- **Statistical features:** mean, variance, and higher moments of packet inter-arrival times or packet sizes, which capture the temporal pattern of traffic.
- **Protocol features:** fractions of traffic belonging to each protocol (UDP, TCP, ICMP) or port group, capturing the composition of the traffic mix.
- **Source concentration features:** metrics such as the fraction of traffic attributable to the top source IP address or the number of distinct sources, which distinguish volumetric attacks (few sources, high volume) from distributed ones (many sources, moderate individual volume).

The choice of features directly determines what patterns the model can learn. Features derived exclusively from packet headers, rather than payload contents, are compatible with encrypted traffic and impose a lower per-packet extraction cost, making them well suited to line-rate deployment. An important practical consideration is that tree-based models operate by comparing individual feature values against thresholds and are therefore invariant to monotonic rescaling: unlike distance-based or neural models, they do not require feature normalisation or standardisation, which simplifies the extraction pipeline and eliminates a potential source of train/test data leakage. The mapping from raw packets to feature vectors is described in detail in Chapter 5.

2.3.3 Ensemble Methods: Bagging and Boosting

A single decision tree partitions the feature space by applying a sequence of binary threshold tests on individual features, assigning a class label to each leaf region. While decision trees are interpretable and fast to evaluate, they suffer from high variance: small changes in the training data can produce very different trees. Ensemble methods address this by combining the predictions of many trees.

- **Bagging** (Bootstrap Aggregating) trains each tree on a different random bootstrap sample of the training data and averages their predictions. The canonical bagging ensemble is the *Random Forest* [14], which additionally randomises the subset of features considered at each split. By decorrelating individual trees, Random Forest substantially reduces variance without increasing bias.
- **Boosting** takes a different approach: trees are trained sequentially, with each tree fitted to the residual errors of the ensemble built so far. The final prediction is a weighted sum of all trees. This sequential correction of errors reduces bias, often achieving lower training error than bagging at the cost of higher sensitivity to noisy labels. Gradient Boosting [14] frames this process as gradient descent in function space, making it straightforward to optimise arbitrary differentiable

loss functions. Modern implementations such as XGBoost, LightGBM, and HistGradientBoosting extend this framework with histogram-based split-finding, regularisation, and native support for missing values.

For structured network traffic data, tree-based ensembles offer several advantages over deep learning alternatives such as recurrent neural networks or transformers [14]: they require no feature scaling, are robust to irrelevant features through gain-based split selection, and converge to competitive accuracy on datasets of typical network monitoring scale (tens of thousands of labelled windows) in seconds rather than hours. Their predictions are also interpretable via feature importance scores derived from split gains, which aids in validating that the model is relying on meaningful traffic statistics rather than dataset artefacts. Section 2.4 describes LightGBM, the gradient boosting implementation that combines these properties with a C inference API suitable for embedded data-plane deployment.

2.4 LightGBM and Embedded Inference

LightGBM [15] is a gradient boosting framework developed by Microsoft Research that implements the gradient boosted decision tree (GBDT) algorithm with a set of algorithmic improvements targeting training speed and memory efficiency. Beyond its training-time characteristics, LightGBM provides a self-contained C shared library with a stable C API that allows trained models to be loaded and evaluated inside arbitrary C or C++ programs, without any dependency on Python or the training environment. This property makes it particularly suitable for embedding a trained classifier into a data-plane application.

2.4.1 LightGBM Internals

Standard gradient boosting trains each successive tree by performing an exact greedy search over all feature values to find the split that maximises the gain on the current residuals. For datasets with many features and many samples this scan is computationally expensive and memory-intensive. LightGBM replaces exact split finding with two complementary techniques:

- **Histogram-based split finding.** Each continuous feature is discretised into at most B bins (typically $B = 255$) by computing a histogram of gradient statistics. The optimal split for a given feature is then found by scanning only the B bin boundaries rather than all distinct values, reducing the per-split cost from $O(n)$ to $O(B)$ and the memory requirement from storing sorted feature values to storing two integers per bin per feature.

- **Leaf-wise tree growth.** Most GBDT implementations grow trees level by level, splitting all leaves at a given depth before proceeding to the next. LightGBM instead always expands the leaf with the highest gain, regardless of depth. This asymmetric growth strategy produces deeper, more specialised subtrees that converge to lower training loss in fewer iterations, at the cost of requiring an explicit maximum-depth constraint to prevent overfitting.

LightGBM also implements two data reduction strategies that further reduce training cost without sacrificing accuracy:

- **Gradient-based One-Side Sampling (GOSS).** Instances with large gradients (large residuals) carry the most information for the next tree and are always retained. A random fraction of the remaining well-fitted instances is sampled, reducing the effective dataset size while preserving the gradient distribution seen by each split search.

- **Exclusive Feature Bundling (EFB).** Sparse features that rarely take non-zero values simultaneously are merged into a single dense feature. This reduces the effective number of features and the number of histograms that must be computed and scanned at each split, yielding near-linear speedups on high-dimensional sparse inputs.

Overfitting is controlled through explicit regularisation: L1 (λ_1) and L2 (λ_2) penalties are added to the leaf output values, shrinking small leaf scores towards zero; a minimum child sample count (`min_child_samples`) prevents splits that would create leaves with too few training instances; and a `max_depth` parameter bounds the asymmetric leaf-wise growth described above.

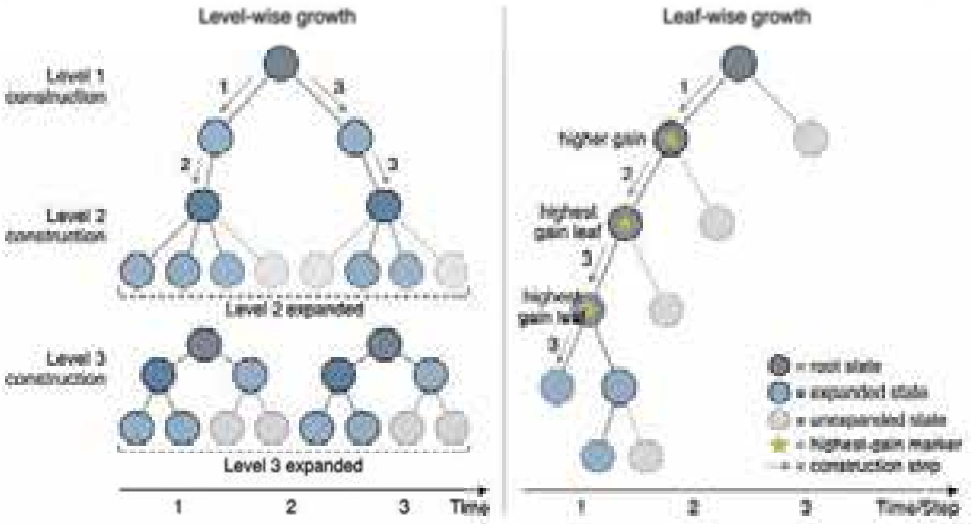

Figure 2.7: Level-wise (left) vs. leaf-wise (right) tree growth in gradient boosting.

The output of a trained LightGBM model for a multi-class problem is a set of K additive ensembles of trees, one per class. Each tree is a binary decision tree whose internal nodes store a feature index and a bin threshold; each leaf stores a scalar output value. At inference time, a feature vector $\mathbf{x}$ is passed through all trees: for each tree, the vector traverses from root to leaf by evaluating binary comparisons, and the corresponding leaf value is accumulated. After summing the contributions of all trees, a softmax transformation converts the K raw scores into a probability distribution over classes, and the class with the highest probability is returned as the prediction.

2.4.2 Model Serialisation and the C Inference API

A trained LightGBM model is serialised to a plain-text file that encodes all tree structures, split thresholds, and leaf values in a human-readable format. The file can be distributed alongside an application or compiled into a binary as a string literal; no training code or Python runtime is required at inference time.

The LightGBM C API, exposed through the `lightgbm.h` header and the `libLightGBM` shared library, provides the functions necessary for inference:

- `LGBM_BoosterLoadModelFromFile()` loads a serialised model from disk and returns an opaque `BoosterHandle` that encapsulates the full tree ensemble.
- `LGBM_BoosterPredictForMat()` accepts a flat row-major matrix of feature values covering one or more samples simultaneously, the number of rows and columns, and a prediction type flag, and writes the output scores directly to a caller-supplied buffer. Batch prediction over multiple samples incurs no additional per-call overhead compared to a single prediction, making it efficient to classify an entire set of windows at the end of a measurement period in one call. No dynamic memory allocation takes place inside the library during inference.
- `LGBM_BoosterFree()` releases the resources associated with a handle when the application terminates.

The inference path, loading the model into a `BoosterHandle` and calling `LGBM_BoosterPredictForMat()`, is *thread-safe*: multiple threads can invoke prediction concurrently on the same handle without any locking, since inference is a read-only traversal of the immutable tree structures. This property is directly relevant to multi-threaded deployments where several worker threads may need to invoke the classifier independently.

The computational cost of a single prediction call is $O(T \cdot D)$, where T is the total number of trees in the ensemble and D is the maximum tree depth. For typical configurations ($T \approx 100$ to 500, $D \leq 10$), each prediction requires a few thousand simple

comparisons and additions, placing the per-call latency in the range of tens of microseconds on a modern CPU core. This cost is incurred once per detection window rather than once per packet, so it does not appear on the per-packet critical path.

2.5 CloudLab: Bare-Metal Research Infrastructure

CloudLab [16] is a scientific infrastructure operated by the University of Utah, in collaboration with the University of Wisconsin and Clemson University. It provides researchers with on-demand access to configurable bare-metal hardware for systems and networking experiments, with an emphasis on reproducibility and experimental isolation.

Unlike public cloud platforms, which expose virtualised resources subject to hypervisor overhead and unpredictable interference from co-located tenants, CloudLab allocates physical servers exclusively to each experimenter. This ensures that performance measurements reflect the true behaviour of hardware and software under study — a property that is essential for reliable network experimentation at line rate. Table 2.2 summarises the principal differences between CloudLab and public cloud deployments from the perspective of systems research.

Table 2.2: CloudLab bare-metal versus public cloud for systems research.

Property	CloudLab	Public Cloud
Node isolation	Physical (exclusive)	Virtual (shared hypervisor)
Kernel control	Full (bare metal)	Limited (guest OS)
NIC hardware access	Direct (SR-IOV, PMD)	Emulated or para-virtual
NUMA topology	Fully exposed	Hypervisor-managed
Network topology	Configurable (L2)	Provider-managed
Measurement reproducibility	High	Low (noisy neighbours)

2.5.1 Bare-Metal Provisioning and Profiles

Experiments in CloudLab are defined through *profiles*, parameterised descriptions of the hardware topology and software environment required for a given experiment. A profile specifies the number and type of physical nodes, the network interconnects between them, and the disk image to be loaded at boot time. Profiles are expressed using RSpec, the resource description format defined by the GENI project [17], or through the Geni-Lib Python library, which generates RSpec programmatically. When an experimenter instantiates a profile, CloudLab's control plane reserves the requested physical machines, loads the specified operating system image onto each node via network boot, and configures the layer-2 topology using programmable switches. The

result is a reproducible, isolated environment that can be recreated on demand — and shared with other researchers — from the same profile definition, supporting independent verification of experimental results.

Nodes are allocated exclusively to a single experiment for its duration. The operating system image is written directly to the node's local disk, so the experimenter has full control over the kernel version, kernel parameters, and installed packages. Persistent storage is available through network attached volumes that survive experiment termination, allowing datasets and build artefacts to be retained across sessions.

2.5.2 Hardware and Network Capabilities

CloudLab offers several hardware clusters, each composed of server nodes with different processor generations, memory capacities, and NIC models. Nodes in the high-speed clusters are typically dual-socket NUMA systems, and CloudLab exposes the full NUMA topology to the experimenter, enabling the kind of NUMA-aware memory and thread placement described in Section 2.1. Of particular relevance for high-speed packet processing research are nodes equipped with multi-port 25 Gbps or 100 Gbps Ethernet adapters from the Mellanox ConnectX series [18]. These NICs support SR-IOV, hardware RSS, and the DPDK Poll-Mode Drivers required for line-rate packet I/O without kernel involvement, making CloudLab a suitable platform for high-throughput network experiments without the need for dedicated laboratory infrastructure.

Inter-node connections are established through programmable Ethernet switches that support VLAN isolation. In some clusters, direct layer-2 links are available between pairs of nodes, providing low-latency connectivity suitable for reproducing the topology of a real network segment. This allows one node to act as a traffic generator and another to run the packet processing application under test, while keeping experimental traffic isolated from the shared management network.

Chapter 3

State of the Art

3.1 DDoS Attack Taxonomy

Distributed Denial of Service attacks encompass a wide variety of techniques that differ in their target network layer, exploitation mechanism, and the resources they aim to exhaust. A systematic taxonomy is a prerequisite for designing a detection system that can distinguish among attack types rather than issuing a single binary alert. The survey by Zargar et al. [19] remains the most widely adopted classification framework, organising DDoS attacks along two orthogonal dimensions: the network layer targeted and the flooding mechanism employed. Industry threat reports [1, 2] confirm that the attack types identified in that taxonomy continue to represent the dominant threat vectors observed in production networks.

3.1.1 Classification by Network Layer

The primary dimension in DDoS taxonomies is the Open Systems Interconnection (OSI) layer at which the attack exhausts resources.

- **Volumetric attacks (L3/L4)** aim to saturate the bandwidth of the link or the packet-processing capacity of the target's infrastructure. They send large volumes of traffic at the network or transport layer, either directly from spoofed source addresses or amplified through third-party reflectors. The metric of damage is packets per second or bits per second, and the attack is effective regardless of the target application.

- **Protocol exploitation attacks (L4)** target finite state machine resources in transport-layer protocol implementations. The canonical example is the TCP SYN flood, in which the attacker sends a large number of SYN packets with

spoofed source addresses, causing the victim to allocate half-open connection state for each packet without ever completing the handshake. Unlike pure volumetric attacks, protocol exploitation attacks can be effective even at moderate packet rates.

- **Application-layer attacks (L7)** direct traffic at a specific service rather than the underlying network. HTTP floods, for example, issue syntactically valid HTTP requests that consume server-side resources (threads, database connections, TLS handshake computation) at a rate that legitimate clients would not produce. Application-layer attacks are harder to distinguish from legitimate traffic at the network layer and require payload inspection or session-level features to detect.

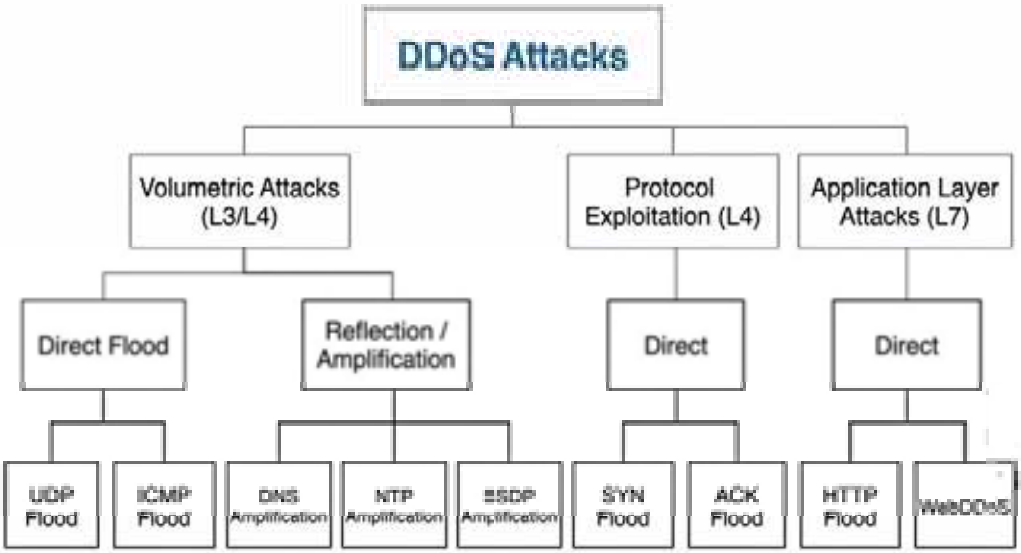

Figure 3.1: DDoS attack taxonomy by network layer and flooding mechanism.

3.1.2 Classification by Flooding Mechanism

Within each layer, attacks are further classified by whether they generate traffic directly or exploit third-party infrastructure as a traffic multiplier.

- **Direct flood.** The attacker's botnet sends packets directly to the victim. Traffic volume is bounded by the aggregate upload bandwidth of the attacking sources. Typical examples are SYN flood, UDP flood, and ICMP flood.

- **Reflection and amplification.** The attacker sends small requests to publicly accessible servers (reflectors) with the victim's IP address forged as the source. The reflectors respond to the victim, reflecting the traffic. If the response is substantially larger than the request, the attack also achieves amplification, multiplying the effective bandwidth by the amplification factor of the exploited protocol.

Many UDP-based services exhibit large amplification factors: NTP monlist responses can be ∼200× the size of the triggering request [20]; DNS ANY queries can yield factors of 28 × to 54 ×; SSDP, LDAP, and TFTP all exceed 30 × [19, 1]. This asymmetry makes reflection amplification the dominant mechanism in large-scale volumetric attacks.

3.1.3 UDP Reflection and Amplification Attacks

Among the attack types catalogued in the CIC-DDoS2019 benchmark [21], UDP reflection and amplification attacks represent the largest and most varied group. Each variant exploits a different UDP-based application protocol whose servers respond to small, spoofable requests with disproportionately large replies. Table 3.1 summarises the principal protocols documented in the literature, together with their standard port and reported amplification factors [20].

Table 3.1: UDP reflection/amplification attack types and amplification factors.

Attack	Port	Amplification factor	Exploited service
DNS	UDP/53	∼28–54×	Domain Name System (ANY query)
NTP	UDP/123	∼200×	Network Time Protocol (monlist)
LDAP/CLDAP	UDP/389	∼56×	Connectionless LDAP
MSSQL	UDP/1434	∼25×	SQL Server Browser
NetBIOS	UDP/137	∼4×	NetBIOS Name Service
Portmap	UDP/111	∼7×	RPCbind / Portmapper
SNMP	UDP/161	∼6×	Simple Network Management Protocol
SSDP	UDP/1900	∼30×	Simple Service Discovery Protocol
TFTP	UDP/69	∼60×	Trivial File Transfer Protocol

The prevalence of UDP as the transport for these attacks is not coincidental. UDP's connectionless nature allows source IP addresses to be trivially spoofed, so the reflector cannot distinguish a legitimate client from an attacker. Additionally, many of these services are deployed on legacy infrastructure without rate-limiting or source-address validation, making them persistently exploitable. Industry telemetry [1, 2] consistently identifies DNS, NTP, and SSDP amplification as the three most frequently observed attack vectors in large-scale volumetric campaigns, a pattern that motivates treating each protocol individually in the classification problem rather than collapsing all UDP amplification into a single class.

3.2 Traditional Detection Approaches

Before machine learning techniques were applied to network security, DDoS detection relied on three broad families of approaches: threshold-based statistical monitoring, signature-based intrusion detection, and flow-based anomaly detection. Each family has been deployed in production networks and remains in use today; however, each also exhibits structural limitations that make it insufficient for the requirements of line-rate, multi-class detection on high-speed encrypted traffic. The survey by Zargar et al. [19] provides a systematic analysis of these limitations in the context of flooding attacks.

3.2.1 Threshold-Based and Statistical Methods

The simplest and historically earliest form of DDoS detection consists of measuring a scalar traffic metric and triggering an alert when that metric exceeds a predefined threshold. Typical metrics include total packets per second, SYN segments per second, the ratio of inbound to outbound bytes, or the number of distinct source IP addresses observed within a fixed time window. Traffic statistics are collected either via SNMP polling of router and switch counters or, at finer granularity, via sampled packet export protocols such as sFlow [22]. Because measurement and comparison are $O(1)$ operations, threshold-based detectors impose negligible computational overhead and are compatible with arbitrarily high link speeds.

Three variants of this approach appear in the literature and in deployed systems:

- **Static thresholds.** A fixed upper bound is configured for each metric based on historical baselines. Any exceedance triggers an alarm. This is the simplest form and the most prone to false positives during legitimate peak traffic.

- **Adaptive thresholds.** The bound is updated dynamically using time-series models such as exponential weighted moving averages or Holt–Winters forecasting [19]. The detector adjusts to diurnal traffic patterns, reducing false positives during predictable load spikes while remaining sensitive to sudden anomalies.

- **Entropy-based detection.** Rather than monitoring a single scalar, entropy-based systems track the Shannon entropy of a traffic distribution, for example the entropy of the source IP address distribution over a one-minute window. A DDoS flood concentrates traffic on a single destination (reducing destination entropy) while a reflection attack generates many spoofed sources (increasing source entropy). Entropy metrics thus capture the structural shift in traffic composition that volume metrics alone cannot detect [19].

Despite these refinements, the fundamental limitation of all threshold-based systems is their inability to distinguish among attack types. A volume threshold fires during a UDP amplification attack and during a SYN flood alike, providing no information about which protocol is being abused or whether the traffic is spoofed. This prevents any targeted mitigation and generates false positives during legitimate traffic bursts, such as flash crowds following major news events or large content distribution spikes [19, 5]. Equally important, low-rate attacks, such as TCP connection exhaustion at moderate packet rates, may remain entirely below a volume-based threshold while still causing service disruption.

3.2.2 Signature-Based Intrusion Detection

Signature-based Intrusion Detection Systems (IDS) and their inline counterpart, Intrusion Prevention Systems (IPS), detect attacks by matching packet contents or connection behaviour against a curated database of known attack patterns. Snort [23] and Suricata [24] are the two most widely deployed open-source implementations of this approach. Each rule specifies a set of conditions, protocol, port range, header flags, and regular expressions over the payload byte sequence, that together characterise a specific attack variant. When an incoming packet satisfies all conditions of a rule, the engine raises an alert or, in IPS mode, drops the offending packet inline. Rule sets are distributed by community organisations such as the Snort Subscriber Rule Set and the Emerging Threats project, and are updated continuously as new attack variants are discovered.

Signature engines have two principal strengths. First, they achieve very high precision against known attacks: a rule that matches an NTP monlist request payload will not fire on legitimate NTP traffic. Second, because rules encode human expert knowledge, they are directly interpretable and can be audited, tuned, and explained to operators without requiring statistical expertise.

However, the approach exhibits three structural weaknesses that limit its applicability at high speeds and in modern encrypted environments:

- **Zero-day blind spot.** Signature engines can only detect attacks whose pattern has been previously catalogued. Novel attack variants, or known attacks with trivially obfuscated payloads, evade detection entirely until a matching rule is authored and distributed.

- **Encryption incompatibility.** Once a session is protected by TLS 1.3 [7] or QUIC [8], the application-layer payload is opaque and no payload-derived rule can fire. The engine is reduced to matching on unencrypted transport-layer fields,

losing the majority of its discriminative capacity.

- **DPI throughput ceiling.** Signature engines must perform deep packet inspection on every packet, which introduces a processing cost that grows with rule-set size and link speed. Throughput benchmarks for Suricata in IPS mode show significant packet drop rates on 10 Gbps links without dedicated hardware offload [24]. On 25 Gbps or 100 Gbps links, kernel-based DPI is not viable at full rate without either sampling, which reduces coverage, or expensive SmartNIC offload.

3.2.3 Flow-Based Anomaly Detection

Flow-based anomaly detection operates at a higher level of aggregation than packet inspection, trading per-packet detail for the ability to observe traffic patterns across thousands of concurrent connections simultaneously. A *flow* is defined as the set of packets sharing the same five-tuple (source IP, destination IP, source port, destination port, protocol) within a time interval. Routers and switches export flow records using Cisco NetFlow [25] or its successor standard IPFIX [26]; each record contains aggregate statistics including packet count, byte count, start and end timestamps, and TCP flag bitmask. A dedicated collector receives these records and applies statistical or ML models to identify anomalies.

The anomaly detection algorithms applied to flow data range from simple statistical tests to more sophisticated models:

- **Volume-based anomaly detection.** CUSUM (Cumulative Sum) and related sequential change-point tests detect abrupt changes in per-protocol byte rates or packet counts, and are widely used in production network monitoring platforms.
- **Clustering and classification.** Flow records are grouped by similarity in the multi-dimensional feature space (packet rate, duration, byte ratio, flag counts), and clusters that deviate significantly from the established baseline are flagged. k-means and DBSCAN have both been applied to this problem [19].
- **Entropy-based flow monitoring.** The Shannon entropy of the source-IP or destination-port distribution computed over a sliding window of flow records captures structural shifts in traffic composition that aggregate volume metrics miss.

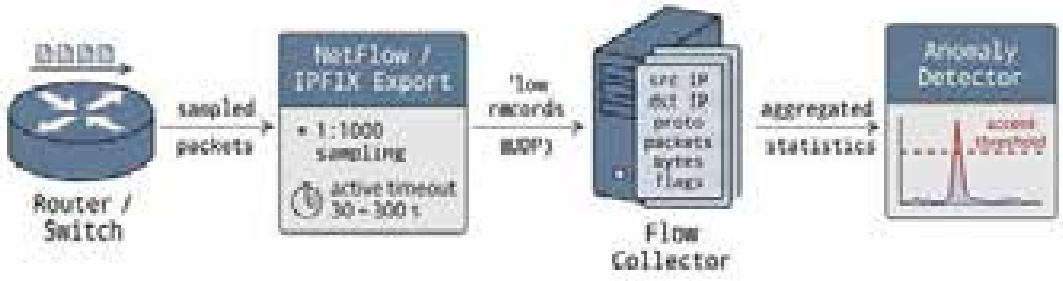

Figure 3.2: NetFlow/IPFIX flow export and anomaly detection pipeline.

Flow-based detection provides richer multi-dimensional visibility than threshold-based approaches and does not require payload access, making it compatible with encrypted traffic. However, it introduces two constraints that are fundamental to the paradigm rather than incidental to any specific implementation. First, a flow record can only be exported after the flow terminates or a keepalive timeout (typically 30 s to 300 s) expires; this bounds detection latency to tens or hundreds of seconds, incompatible with real-time mitigation. Second, flow sampling is routinely applied at high-speed interfaces, one in every 1 000 or 10 000 packets, so low-volume flows may be missed entirely, and the statistics of sampled records are approximations that systematically degrade detection accuracy for low-rate or short-lived attacks [19].

3.2.4 Comparison of Traditional Approaches

Table 3.2 compares the three families across six dimensions relevant to inline DDoS detection. The pattern is consistent: no single approach satisfies all requirements simultaneously. Threshold-based systems are fast and lightweight but provide no classification granularity and miss low-rate attacks. Signature engines classify accurately but hit a hard ceiling at high link speeds and are blind to encrypted traffic. Flow-based detectors offer rich multi-dimensional features but introduce latencies that are orders of magnitude above the sub-second response requirement.

Table 3.2: Comparison of traditional DDoS detection approaches.

Approach	Line-rate	Multi-class	Encrypted	Detection latency	Per-packet cost	Low-rate attacks
Threshold-based	Yes	No	Yes	<1 s	Very low	No
Signature (DPI)	No	Yes	No	<100 ms	High	Partial
Flow-based	Yes (sampled)	Partial	Yes	30 s to 300 s	Low	No

The complementary failure modes of these three families motivated the development of machine learning-based detection systems, which aim to combine the classification granularity of signature-based methods with the line-rate scalability of statistical approaches, while avoiding the latency penalty of flow-level aggregation. Section 3.3 surveys this line of work.

3.3 ML-Based Detection Systems

The limitations of threshold and signature-based systems have motivated a growing body of work that applies machine learning to network traffic classification. ML-based detectors replace hand-crafted rules with data-driven models that learn discriminative patterns from labelled traffic traces, enabling multi-class identification of attack types rather than a binary alert. The survey by Ferrag et al. [27] provides a comprehensive review of deep learning approaches for network intrusion detection, and the CIC-DDoS2019 benchmark [21] has become the standard dataset against which most recent systems are evaluated.

3.3.1 Feature Engineering for Network Traffic

The accuracy of any ML-based detector is directly constrained by the feature representation fed to the model. Unlike image or text classification, where the raw input is directly consumable, network traffic must first be transformed into a structured observation vector before a model can be applied. The dominant paradigm in the literature derives features at the *flow level*: all packets belonging to the same five-tuple are aggregated into a single observation vector once the flow has terminated or a timeout has elapsed.

Tools such as CICFlowMeter, used to generate the CIC-DDoS2019 dataset [21], extract up to 80 statistical features per flow. These features fall into several broad categories, each targeting a different behavioural dimension of the traffic:

- **Volume and rate features.** Total packet and byte counts, packets per second, bytes per second, and asymmetry ratios between the forward and backward directions of the flow. Volumetric attacks produce characteristic spikes in these metrics.
- **Temporal features.** Inter-arrival time (IAT) statistics, minimum, maximum, mean, standard deviation, and variance, both per flow and per direction. The IAT distribution distinguishes high-rate floods from connection-oriented traffic with natural pacing.
- **Packet size features.** Forward and backward payload length statistics. Reflection attacks typically exhibit very small requests and very large responses, producing a distinctive asymmetry in these features.
- **Protocol state features.** TCP flag counts (SYN, ACK, FIN, RST, PSH) per direction, window size, and connection completion ratio. SYN floods leave characteristic incomplete handshakes visible in flag statistics.

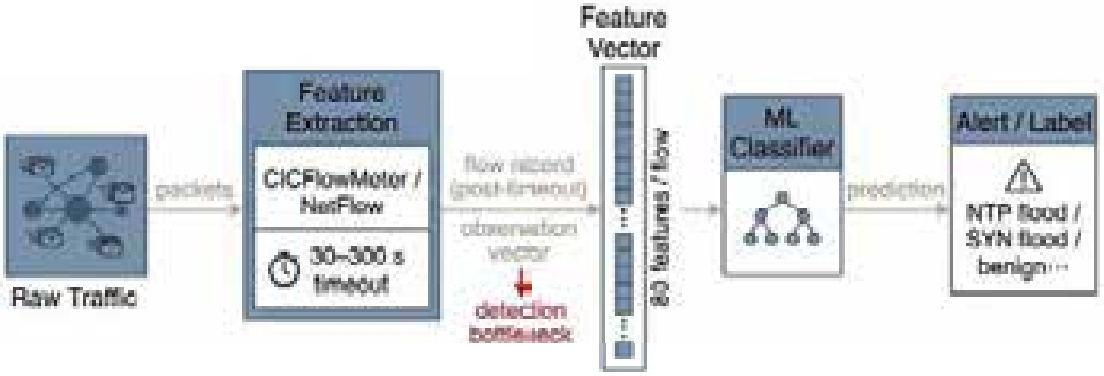

Figure 3.3: ML-based DDoS detection pipeline.

A practical challenge in building classifiers on these datasets is class imbalance: benign traffic and common attack types dominate the sample count, while less frequent attack classes contribute far fewer examples. Oversampling techniques such as SMOTE [27] and cost-sensitive learning are commonly applied to prevent the model from trivially predicting the majority class.

The fundamental constraint of flow-level features is temporal: a flow record can only be computed after the flow has terminated or a keepalive timeout has elapsed, intro-

during the same latency as flow-based detection systems described in Section 3.2.3. An alternative is to derive features from fixed-length time windows rather than individual flows, accumulating packet-level statistics over a sliding or tumbling interval. Sketch-based feature extraction, discussed in Section 3.5, is the primary technique for achieving window-based feature computation at line rate.

3.3.2 Classical Machine Learning Approaches

Among classical ML methods, tree-based ensemble models have consistently achieved the highest reported accuracy on flow-level DDoS detection benchmarks, largely because their non-parametric, non-linear decision boundaries are well-suited to the heterogeneous feature distributions produced by CICFlowMeter. The main model families applied in the literature are:

- **Random Forests [14].** An ensemble of independently trained decision trees that vote on the final label. The randomised feature subsetting at each split reduces variance and prevents overfitting. On the CIC-DDoS2019 dataset, Random Forest classifiers regularly report macro-averaged F1 scores above 0.97 across all attack classes [27, 21]. A notable practical advantage is the Gini impurity-based feature importance score, which allows practitioners to identify the most discriminative features and prune the vector to reduce inference cost.

- **Gradient Boosting (XGBoost, LightGBM).** Boosted tree ensembles that train sequentially, with each tree correcting the residuals of its predecessors. LightGBM [15] in particular achieves inference times of less than 1 ms per observation even for large ensembles, making it attractive for latency-sensitive deployments. On benchmark datasets it matches or slightly exceeds Random Forest accuracy.

- **Support Vector Machines (SVM).** SVMs find the maximum-margin hyperplane in a kernel-projected feature space and have been applied to binary (benign vs. attack) classification with good precision. However, their inference cost grows with the number of support vectors and they scale poorly to multi-class settings with many attack types and millions of training samples [27].

The principal limitation shared by all these systems is that they inherit the flow-expiration latency of their feature pipeline: the model can only produce a classification after the flow record has been computed, which requires the flow to terminate or a timeout to fire. Inference itself is fast, sub-millisecond for tree-based models, but the end-to-end detection latency is dominated by the feature extraction stage, making real-time inline deployment infeasible.

3.3.3 Deep Learning Approaches

Deep learning methods have been extensively explored for network traffic classification, motivated by their ability to learn hierarchical feature representations without manual feature engineering. Three architectural families dominate the literature:

- **Convolutional Neural Networks (CNNs).** CNNs treat a matrix of per-packet header fields, arranged as rows, one per packet in a flow, as a two-dimensional input analogous to an image. Convolutional filters learn local patterns across adjacent packets, capturing short-range temporal correlations in traffic behaviour. They achieve competitive F1 scores on CIC-DDoS2019 but require the same flow-level feature collection as classical models [27].

- **Long Short-Term Memory (LSTM) networks [28].** LSTMs explicitly model the sequential dependency between successive packets or windows, making them well suited to capturing the temporal evolution of an attack, for example the escalating rate of a SYN flood in its early seconds. Their gated memory cells allow long-range dependencies to be retained across hundreds of time steps, at the cost of significantly higher inference complexity than feedforward models.

- **Autoencoders and anomaly-based deep detectors.** Unsupervised autoencoders trained on benign traffic learn a compact representation of normal behaviour; inputs with high reconstruction error are flagged as anomalous. This approach does not require labelled attack traffic for training but is limited to binary detection and cannot classify the attack type [27].

While deep models occasionally report marginal F1 improvements over Random Forest on minority attack classes, their practical drawback is inference cost. A forward pass through a multi-layer LSTM involves matrix multiplications proportional to the hidden state size, resulting in inference latencies of tens to hundreds of milliseconds per batch even on GPU hardware [27]. Because a line-rate detection system must run on the same CPU cores handling packet I/O, GPU acceleration is unavailable on the critical path, and the inference overhead of deep models becomes a hard constraint rather than an engineering concern.

3.3.4 Latency as the Fundamental Constraint

A common thread across the surveyed ML-based systems is that their end-to-end detection latency is dominated not by model inference, which is sub-millisecond for tree-based models and at most tens of milliseconds for deep networks, but by the time required to produce the feature vector. This distinction is critical: even if a perfectly

accurate model were available, it could not reduce detection latency below the flow-expiration timeout of its feature extractor.

A concrete example is MULTI-LF [6], a continuous learning framework designed to detect malicious traffic across multiple network environments. MULTI-LF combines flow-level and payload-derived features with an online learning component that adapts the model to concept drift, achieving macro-averaged F1 scores above 0.99. Despite this high accuracy, the system reports end-to-end detection latencies exceeding 800 ms, the majority of which is spent waiting for flow records to be computed. As noted in Chapter 1, such latencies are unacceptable in environments where an undetected volumetric attack can saturate a high-speed link within hundreds of milliseconds of onset.

The accuracy–latency trade-off observed across all surveyed systems is not a consequence of engineering choices that could be optimised away: it is structural. As long as the feature pipeline requires flow termination or a keepalive timeout, detection latency will be bounded below by that timeout regardless of the model used. Table 3.3 quantifies this pattern across representative systems.

Table 3.3: Comparison of representative ML-based DDoS detection systems.

System	Model	Feature source	Feature latency	Inference latency	F1 (macro)	Encrypted
Ferrag et al. [27]	Random Forest	Flow-level (CI-CFlowMeter)	30 s to 300 s	<1 ms	>0.97	Yes
Ferrag et al. [27]	CNN	Flow-level	30 s to 300 s	1 ms to 10 ms	>0.96	Yes
Ferrag et al. [27]	LSTM	Packet sequences	5 s to 30 s	10 ms to 100 ms	>0.95	Yes
MULTI-LF [6]	Online ensemble	Flow + payload	>700 ms	<100 ms	>0.99	No

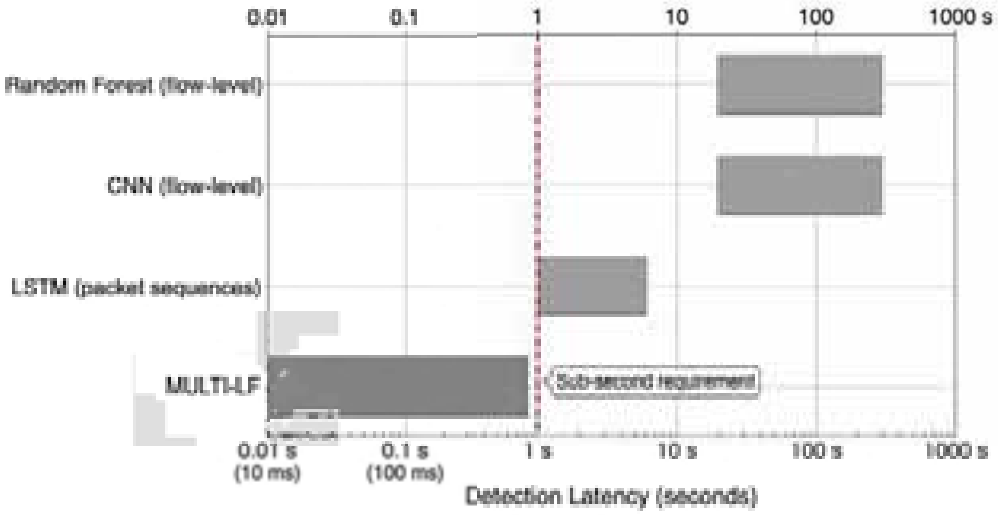

Figure 3.4: Detection latency comparison across ML-based DDoS detection systems.

3.4 High-Speed Packet Processing Solutions

The limitations of the kernel networking stack identified in Chapter 1 motivate the use of specialised packet processing frameworks that bypass or extend the kernel to achieve line-rate throughput. Three architectural approaches have seen broad adoption: kernel-bypass user-space I/O via DPDK, in-kernel programmable packet processing via eBPF/XDP, and hardware offload to programmable SmartNICs. Each is contrasted in Section 3.4.4 against the traditional kernel stack as a performance baseline, and each makes different trade-offs between throughput, programmability, and operational complexity.

3.4.1 Kernel Bypass: DPDK

The Data Plane Development Kit (DPDK) [3] is the reference framework for kernel-bypass packet processing. Its architecture and programming model are described in detail in Chapter 2; this section positions DPDK in the context of related approaches.

DPDK achieves its performance by entirely removing the kernel from the data path. A Poll Mode Driver (PMD) maps the NIC's descriptor rings directly into user-space memory, and the application polls these rings continuously without invoking system calls or handling interrupts. Combined with the use of hugepages to eliminate TLB pressure and Receive Side Scaling (RSS) to distribute flows across cores, DPDK can sustain throughputs of 40 Gbps and above on commodity servers [3]. The framework provides a rich library of data structures optimised for packet processing, including lock-free ring buffers, memory pools, and exact-match and longest-prefix-match tables.

The key mechanisms that underpin DPDK's line-rate performance can be summarised as follows:

- **Zero-copy packet access.** In the kernel networking stack, every received packet is copied from DMA-mapped NIC memory into a kernel `sk_buff`, and again into user space when the application reads a socket. DPDK eliminates both copies: the PMD DMA-maps *mbuf* buffers from a pre-allocated memory pool (`rte_mempool`) directly into NIC descriptor rings. The application receives a pointer to the mbuf, reads or modifies the packet in place, and returns the buffer to the pool after processing, with zero data copies on the receive path.

- **NUMA-aware memory allocation.** On multi-socket servers, accessing memory on a remote NUMA node incurs an additional memory latency penalty of 40–80 ns compared to local DRAM access. DPDK's `rte_mempool` API allocates hugepage memory on the NUMA node local to the CPU core that will process the packets. NIC queues are then pinned to cores on the same NUMA node as the NIC's PCIe endpoint, ensuring that all DMA transfers and descriptor updates remain within the same NUMA domain.

- **Dedicated polling cores (lcore model).** DPDK assigns each worker thread to a dedicated logical core (lcore). The lcore spins in a tight loop polling the NIC's receive ring, consuming 100% of its CPU budget at all times. This eliminates the scheduling jitter and interrupt latency that affect general-purpose applications. On a 25 Gbps link with 64-byte minimum-size packets, the arrival rate is approximately 37 Mpps; dedicating two to three cores to polling ensures that no descriptors are left unserviced.

- **Batch processing.** The PMD burst API retrieves up to 32 packets per poll iteration, amortising the overhead of pointer dereferences, branch mispredictions, and cache-line loads across the batch. Burst sizes of 16–32 are empirically optimal for most workloads, yielding a significant improvement in per-packet processing cost compared to a one-at-a-time polling model.

The principal cost of DPDK is the dedication of one or more CPU cores to polling. Because the application loops continuously on the descriptor rings, those cores cannot be used for other tasks, and the operatingsystem scheduler is effectively excluded from the data path. On multi-core platforms this is acceptable, as cores are provisioned specifically for packet processing; however, it makes DPDK unsuitable for workloads that require sharing CPU resources with general-purpose server tasks. DPDK also requires NIC hardware supported by a PMD, which restricts portability to a well-defined set of vendor-specific drivers.

3.4.2 eBPF and XDP

Extended Berkeley Packet Filter (eBPF) is a Linux kernel subsystem that allows verified, sandboxed programmes to be executed within the kernel at specific hook points [29]. The eXpress Data Path (XDP) hook is the earliest point at which an eBPF programme can process a packet: it executes within the NIC driver, before the packet is handed to the kernel's network stack and before any memory allocation for a socket buffer (`sk_buff`) takes place. This placement allows XDP programmes to make forwarding, filtering, or redirect decisions at rates comparable to dedicated hardware, with measured throughputs of 24 Mpps on a single core for simple drop programmes [29].

XDP supports three operational modes that trade performance against hardware requirements:

- **Native XDP (driver mode).** The XDP programme is loaded directly into the NIC driver and executes as soon as the DMA engine places a packet into a descriptor ring, before any kernel data structure is allocated. This is the highest-performance mode and is supported by a growing set of drivers (mlx5, i40e, ixgbe). Throughput is bounded by NIC and PCIe bandwidth rather than CPU cycles.

- **Generic XDP (skb mode).** A software fallback path that inserts the XDP hook after the kernel has already allocated an `sk_buff`. It is compatible with any NIC driver but foregoes most of the performance advantage, since the expensive skb allocation has already occurred. Generic mode is primarily used for development and testing on unsupported hardware.

- **Offloaded XDP.** On supporting SmartNICs (e.g., Netronome Agilio), the XDP programme is compiled directly onto the NIC's embedded processor and executes without consuming any host CPU cycles. This is the highest-performance tier but requires vendor-specific toolchains and imposes the tightest constraints on programme complexity.

The XDP programme communicates its forwarding decision to the driver via return codes: `XDP_DROP` discards the packet at the driver with minimal overhead; `XDP_PASS` hands it to the normal kernel stack; `XDP_TX` hairpins it back out the same interface; and `XDP_REDIRECT` forwards it to another interface or to a user-space process via an AF_XDP socket. The AF_XDP path achieves near-zero-copy delivery to user space by mapping the NIC's UMEM region directly into the application's address space, avoiding the per-packet copy that a conventional `recvmsg()` call would perform. This model has been used in production DDoS mitigation deployments: Cloudflare's Katran load

balancer and Cilium's network security platform both use XDP for high-speed packet filtering and forwarding [29].

Despite its performance advantages over the traditional kernel stack, XDP faces structural limitations for stateful, memory-intensive workloads. The *eBPF verifier*, the in-kernel component that analyses every programme before loading, enforces a series of safety properties: all branches must be statically deterministic, unbounded loops are rejected, and the total number of instructions is bounded (4,096 instructions per programme prior to Linux 5.2; raised to one million with bounded loops in later kernels). This prevents the verifier from having to perform halting-problem analysis at load time, but it also restricts the complexity of per-packet computation that can be expressed within a single programme. Maintaining large shared data structures, such as per-core counter arrays or hash tables with many thousands of entries, requires eBPF maps that involve atomic operations and are subject to size limits that vary by kernel version. Finally, XDP programmes execute in the context of the softirq handler and compete with the kernel's interrupt processing, rather than benefiting from the dedicated core isolation that DPDK provides. These constraints make XDP well-suited to stateless filtering tasks, packet dropping, load balancing, and simple forwarding, but structurally less appropriate for stateful, memory-intensive per-packet aggregation workloads on multi-gigabit links.

3.4.3 Programmable SmartNICs

SmartNIC offload represents the highest performance tier in the packet processing hierarchy: programmable ASICs or FPGAs embedded in the NIC can process packets at wire speed without consuming any host CPU cycles. Unlike a conventional NIC that implements a fixed function (descriptor management, DMA, checksum offload), a SmartNIC exposes a general-purpose programmable pipeline that can execute arbitrary per-packet logic before the host CPU is ever involved.

Two classes of SmartNIC architecture are relevant to network security:

- **FPGA-based SmartNICs.** Field-Programmable Gate Arrays provide the highest raw throughput and the lowest and most deterministic per-packet latency, as the processing pipeline is compiled directly into logic gates. Custom parsers, hash functions, and counter arrays can be implemented with sub-100 ns latency. The trade-off is development complexity: FPGA designs require HDL programming (VHDL or Verilog) or high-level synthesis tools, and the compile-test cycle is long.

- **ARM/DPU-based SmartNICs.** Data Processing Unit (DPU) platforms such

as the NVIDIA BlueField series embed a multi-core ARM processor alongside the NIC's ASIC. The ARM cores run a full Linux environment and can execute standard C/C++ applications, making the programming model significantly more accessible. The ASIC provides hardware offload for specific operations (RSS, checksum, tunnel encapsulation/decapsulation, flow table lookups), while the ARM cores handle the control plane and more complex data-plane logic. Performance is lower than FPGA-based designs but still far exceeds host CPU throughput for packet-processing tasks.

Programmable NICs also support P4, a domain-specific language for expressing packet-processing pipelines in terms of parser graphs, match-action tables, and control flow. P4 programmes are compiled to the target architecture (FPGA, ASIC, or software switch), offering a degree of portability across vendors. However, P4's stateless match-action model is poorly suited to stateful traffic aggregation: maintaining running counters, updating hash-indexed arrays, and computing per-window statistics require register primitives and extern functions that are vendor-specific and not standardised across P4 targets.

SmartNIC programmability is therefore constrained by the instruction set and memory architecture of the embedded processor. Complex stateful data-plane applications that require large shared memory structures and dynamic control flow remain difficult to express within current P4 or DPU programming models, increasing development complexity and reducing portability. These constraints position SmartNICs as a compelling platform for offloading simple stateless operations, ACL enforcement, load balancing, tunnel termination, but less mature for general stateful traffic analysis workloads.

3.4.4 Comparison of High-Speed Processing Approaches

Figure 3.5 illustrates the data path each approach takes from NIC to application, and Table 3.4 summarises all four approaches along the dimensions most relevant to inline DDoS detection.

Table 3.4: Comparison of high-speed packet processing approaches for inline DDoS detection.

Approach	Max throughput	Latency	Programmability	CPU cost	Portability
Kernel stack	~5 Mpps	~10 µs	High	Low (IRQ)	Any NIC
eBPF/XDP	~24 Mpps	~1 µs	Medium	Low	Most NICs
DPDK	>100 Mpps	<1 µs	High	Dedicated cores	PMD-supported NICs
SmartNIC (DPU)	>200 Mpps	<100 ns	Medium	None (offloaded)	Vendor-specific

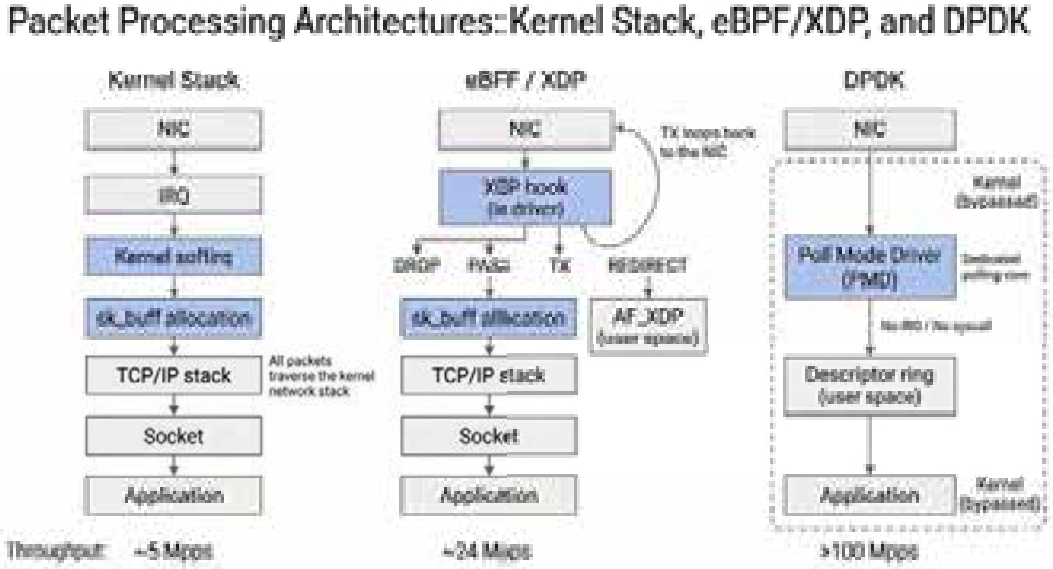

Figure 3.5: Packet processing architectures: kernel stack, eBPF/XDP, and DPDK, showing the data path from NIC to application and the corresponding maximum throughput.

The four approaches span three orders of magnitude in maximum throughput and represent a clear hierarchy of performance versus portability. For inline DDoS detection workloads that combine high packet rates with stateful per-packet computation, DPDK's dedicated-core model and zero-copy memory architecture provide the most favourable trade-off among the surveyed alternatives. The design and implementation of such a system are described in Chapter 5.

3.5 Feature Extraction in Network Intrusion Detection

Feature extraction is the bridge between raw packet streams and the input vectors consumed by a machine learning model. The design choices made at this stage determine not only classification accuracy but also the latency, memory footprint, and hardware requirements of the entire detection pipeline. Two orthogonal dimensions characterise any feature extraction approach:

- **Temporal granularity,** whether features are computed over a complete flow (all packets sharing the same five-tuple, from the first packet until termination or timeout) or over a fixed-size time window (a tumbling or sliding interval that is independent of flow boundaries). Per-flow granularity maximises the statistical richness of each observation but couples detection latency to the flow lifetime. Per-window granularity bounds observation latency to the window duration, at the cost of partial flow statistics.

- **Inspection depth,** whether the extractor reads only the packet header fields available at the transport layer (IP addresses, ports, protocol, TCP flags, packet sizes) or also parses the application-layer payload to derive protocol-specific features. Payload-aware extraction provides richer discriminative features but is incompatible with encrypted traffic and requires DPI hardware or software.

Figure 3.6 maps representative systems onto this two-dimensional design space. The dominant quadrant, flow-level and payload-aware, contains CICFlowMeter and the signature engines Snort/Suricata: tools that accumulate per-five-tuple statistics while inspecting payload content.

The flow-level, header-only quadrant is occupied by NetFlow/IPFIX-based anomaly detectors, which avoid payload access but retain the flow-expiration latency. Kitsune [27], an online ML framework for network intrusion detection, operates per-window on payload-derived features, placing it in the top-right quadrant.

The bottom-right quadrant, window-level and header-only, remains largely unpopulated in the surveyed literature, representing the design space that combines bounded observation latency with encryption compatibility.

Figure 3.6: Feature extraction design space.

3.5.1 DPI-Based Feature Extraction

Deep Packet Inspection based feature extraction is the dominant approach in the academic literature, largely because the CIC-DDoS2019 benchmark [21] was built using CICFlowMeter, a tool that extracts up to 80 statistical features per five-tuple flow by inspecting both packet headers and payloads. Table 3.5 lists the main feature groups that CICFlowMeter produces, covering inter-arrival times, packet length distributions, TCP flag ratios, and protocol-specific payload fields.

Table 3.5: CICFlowMeter feature groups used in flow-level DDoS detection.

Feature group	Count	Description
Packet length statistics	14	Min, max, mean, std of forward/backward packet sizes
Inter-arrival times (IAT)	10	Min, max, mean, std, variance of flow and per-direction IATs
Packet counts and rates	8	Total, forward, backward packet counts; packets per second
Byte counts and rates	6	Total bytes; bytes per second; average bytes per bulk
TCP flag ratios	8	SYN, ACK, FIN, RST, PSH, URG flag counts per direction
Flow duration and timing	4	Flow duration; active/idle time statistics
Bulk and subflow features	12	Bulk packet rate; subflow average packet/byte counts
Header length fields	8	Forward/backward header sizes; window size statistics
Protocol-specific fields	10	DNS query types; HTTP method counts; TLS record sizes
Total	**80**	

DPI based extraction is not a single monolithic tool but a family of implementations that share the same per-flow aggregation model. CICFlowMeter [21] is the canonical offline processor: it reads a packet capture (PCAP) file, reconstructs flows, and outputs a CSV of feature vectors as a post-processing step. For inline deployment, libraries such as nDPI and tools such as Suricata operate in streaming mode, maintaining a per-flow state machine updated on every arriving packet. In both cases, the central data structure is a hash table indexed by the five-tuple, which accumulates running statistics (counts, sums, min/max) until the flow terminates or a timeout fires and the record is emitted for classification. This architecture achieves high discriminative power because payload-derived features carry protocol-specific information that is invisible at the header level. However, it suffers from three structural limitations that are intrinsic to the DPI paradigm:

- **Unbounded per-flow state.** Maintaining a five-tuple hash table that grows with the number of active flows introduces a memory footprint proportional to the number of concurrent connections. On high-speed backbone links, flow table sizes routinely reach tens of millions of concurrent entries [19], and under a SYN-flood or reflection attack the table can grow by millions of spurious half-open entries per second, exhausting available memory or forcing aggressive eviction that degrades feature quality for legitimate flows.

- **Encryption incompatibility.** TLS 1.3 [7] and QUIC [8] encrypt the entire application-layer payload, including DNS-over-HTTPS, HTTP/3, and most modern web traffic. Once traffic is encrypted, any feature that requires reading above the transport header becomes unavailable, and the detector must fall back to the subset of header-only features, losing most of its discriminative capacity.

- **Flow-expiration latency.** A flow record can only be produced after the flow terminates or a keepalive timeout fires. Default NetFlow and CICFlowMeter configurations use timeouts of 30 to 300 seconds, and even aggressive active-timeout settings of 15 seconds still introduce detection latencies that are incompatible with real-time mitigation. Measurement studies report that CICFlowMeter's offline processing throughput is in the range of hundreds of megabits per second [21], making it unsuitable as an inline component on multi-gigabit links.

The three structural limitations identified above represent fundamental constraints of the DPI paradigm that cannot be resolved by engineering optimisations alone. Table 3.6 summarises them across the dimensions most critical for inline, high-speed DDoS detection.

Table 3.6: Structural limitations of DPI-based feature extraction for inline DDoS detection.

Limitation	Root cause	Impact on detection
Unbounded per-flow state	Five-tuple hash table grows with active flow count	Memory exhaustion or aggressive eviction on high-speed backbone links
Encryption incompatibility	TLS 1.3/QUIC encrypt payload above transport layer	Feature set collapses to header-only subset; discriminative power degrades significantly
Flow-expiration latency	Flow records require flow termination or keepalive timeout (30–300 s)	Detection latency orders of magnitude above sub-second mitigation requirement

Taken together, these limitations indicate that DPI-based feature extraction, despite its high discriminative accuracy on benchmark datasets, faces significant engineering challenges when deployed in a line-rate, real-time DDoS detector operating on modern encrypted traffic. Each limitation is addressable in principle, but addressing all three simultaneously within a single inline pipeline requires rethinking the underlying data structures and observation model. A feature extraction architecture that avoids payload inspection, operates on fixed-size data structures, and produces observations on a

sub-second cadence would fill the gap identified in Figure 3.6. The design of such an architecture is the subject of Chapter 5.

Chapter 4

Work Definition

4.1 Motivation

The limitations identified in Chapter 1 and the survey of existing approaches in Chapter 3 motivate the design of a detection system built around three complementary technologies: kernel-bypass packet processing, probabilistic data structures, and embedded machine learning inference.

4.1.1 Kernel-Bypass Packet Processing

The Data Plane Development Kit (DPDK) [3] addresses the throughput bottleneck of the Linux kernel by moving packet processing entirely to user space. By combining poll-mode drivers, hugepage memory allocation, and CPU pinning, DPDK enables a single multi-core server to sustain packet processing rates in the tens of millions of packets per second, sufficient to keep up with high-speed network links without dropping traffic. This makes DPDK a natural foundation for any inline detection system that must inspect every packet in real time.

4.1.2 Probabilistic Data Structures for Traffic Summarisation

Processing packets at line rate leaves very little time per packet for complex computations. Maintaining exact per-IP counters using conventional data structures would require large hash tables that do not fit in CPU cache, leading to frequent cache misses and degraded throughput. Probabilistic sketches such as the Count-Min Sketch [12] and its successors offer a compelling alternative: they summarise traffic statistics in a fixed, small memory footprint with $O(1)$ update and query operations, at the cost of a bounded and controllable estimation error. In this work we adopt OctoSketch [13], a multi-core monitoring framework that applies continuous, change-based aggregation

across worker cores to maintain an accurate global sketch without the accuracy loss of periodic full merges.

Crucially, sketch-based features are derived exclusively from packet headers (source IP, destination port, protocol, and packet size) without inspecting the payload. This property, which we refer to as the *sketch-based advanced* (Sketch-ADV) feature mode, means that the approach remains fully applicable to encrypted traffic, where DPI-based features are unavailable.

4.1.3 Embedded Machine Learning Classification

Threshold-based rules are fast but coarse: they can signal that an anomaly is occurring, but cannot reliably distinguish between different attack types, nor adapt to traffic patterns that fall below fixed thresholds. Machine learning classifiers, by contrast, can learn the statistical signatures of different attack categories from labelled training data and generalize to new traffic conditions. The key challenge is integrating ML inference into a high-speed packet processing pipeline without introducing prohibitive latency. By using the LightGBM C API [15] directly within the DPDK coordinator thread, inference can be performed on a feature vector extracted every few seconds with a latency of just a few milliseconds, orders of magnitude faster than systems that offload classification to a separate process.

4.1.4 The Case for Combining the Three

Neither DPDK alone, nor sketches alone, nor ML alone solves the full problem. DPDK provides the throughput; sketches provide the memory-efficient, encryption-agnostic traffic summaries; and ML provides the multi-class attack classification capability. Together, they form a system that can process traffic at line rate, maintain compact per-IP statistics without payload access, and deliver accurate attack classification, all within a single inline deployment on commodity server hardware.

4.2 Objectives

The primary objective of this work is to design, implement, and evaluate a line-rate ML-based DDoS detection system capable of operating in high-speed and partially encrypted network environments while maintaining competitive classification performance.

This primary objective is decomposed into the following specific objectives:

1. Design a scalable multi-core DPDK-based packet processing architecture capable of sustained line-rate operation without hardware-level packet loss.

2. Implement probabilistic traffic monitoring using sketch-based data structures to efficiently summarise high-cardinality traffic patterns with bounded memory usage.

3. Engineer structured traffic features derived from both payload-aware and header-only probabilistic representations, enabling direct comparison between the two approaches.

4. Demonstrate that header-only Sketch-ADV features achieve multi-class DDoS classification accuracy comparable to feature sets that include DPI-derived payload features.

5. Integrate model inference directly into the packet processing pipeline, ensuring real-time detection without external processing services.

6. Validate the system in a controlled multi-node testbed environment under reproducible traffic generation scenarios.

4.3 Methodology

This section describes the methodological framework used to design, implement, and evaluate the proposed detection system. The methodology integrates high-performance packet processing, probabilistic traffic monitoring, structured feature engineering, and embedded machine learning inference into a unified experimental framework.

4.3.1 Experimental Setup

Experiments are conducted in a controlled multi-node deployment on CloudLab [16], designed to reproduce realistic high-speed network conditions. The testbed comprises a traffic generation node, a monitoring node running the DPDK-based detector, and a victim server, interconnected through high-speed links. This infrastructure enables reproducible experimentation and controlled dataset generation under conditions that reflect the traffic rates and attack patterns of production networks.

The monitoring node is equipped with a multi-core CPU and a high-speed NIC with DPDK-compatible drivers, with hugepage-enabled memory allocation to support the DPDK memory model. All packet processing cores are isolated from the operating system scheduler to guarantee deterministic polling latency.

4.3.2 Feature Extraction Strategies

The system supports multiple feature extraction configurations to enable a direct comparison between payload-dependent and header-only approaches:

- **DPI-Sketch.** Combines features derived from deep packet inspection with global sketch features. This configuration requires access to packet payloads and is therefore inapplicable to encrypted traffic.
- **Sketch-ADV.** Derived exclusively from packet headers, extending the global sketch with per-protocol sketch instances and packet size statistics. This configuration requires no payload access and remains applicable to encrypted traffic.

This design enables direct comparison between payload-dependent and header-only approaches, allowing evaluation of their relative classification performance and operational compatibility with encrypted traffic.

4.3.3 Dataset Generation and Model Training

Detection operates over fixed time windows of 50 ms. For each window, cumulative traffic statistics are aggregated from the sketch data structures and transformed into structured feature vectors. Datasets are constructed by aggregating multiple experimental runs covering different traffic conditions, following a baseline–attack–recovery protocol that enables accurate labelling of each detection window.

Feature normalisation is applied using standard scaling techniques to ensure consistent model training across runs. A LightGBM [15] classifier is trained for supervised multi-class classification, distinguishing between benign traffic and the DDoS attack categories present in the dataset.

4.3.4 Embedded Inference and Detection Logic

The trained model is exported and loaded directly into the DPDK-based detector through the LightGBM C API [15]. Inference is executed once per detection window on a dedicated coordinator core, ensuring real-time classification without inter-process communication or off-path processing services. Classification decisions are determined exclusively by the trained model; statistical threshold counters remain available for monitoring and diagnostic purposes but do not influence alert decisions.

4.4 Planning

The project was structured into seven sequential work blocks, developed incrementally from September through March. Figure 4.1 summarises the timeline. The progression reflects a gradual transition from conceptual foundations to system implementation, experimental validation, and final consolidation.

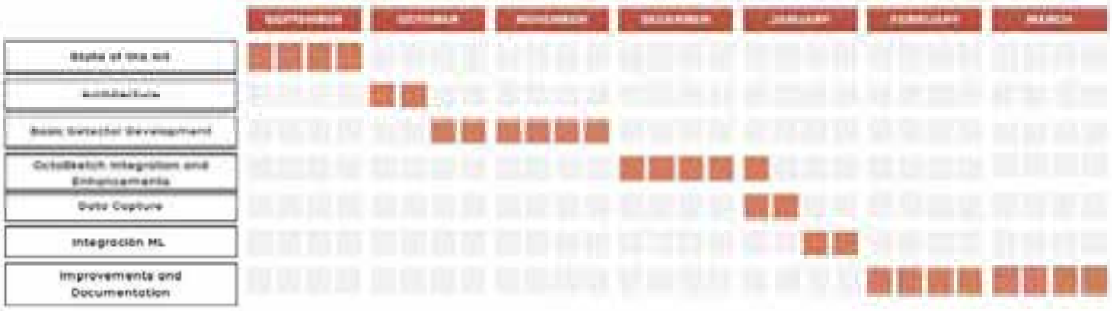

Figure 4.1: Project planning timeline from September to March, structured into seven work blocks.

Block A – State of the Art (September). This initial phase focused on a comprehensive literature review covering DDoS detection approaches, kernel-bypass frameworks, probabilistic data structures, and ML-based classification in high-speed network environments. This stage established the theoretical and methodological foundation of the project.

Block B – Architecture Design (October). The multi-core DPDK-based packet processing architecture was designed during this phase. This included the definition of RSS-based traffic distribution, per-worker data structure organisation, and the coordination logic required for feature aggregation and classification.

Block C – Basic Detector Development (October–November). The core detection pipeline was implemented, including packet parsing, protocol-level counters, and the initial monitoring framework. This phase established the functional baseline of the system before the integration of probabilistic structures.

Block D – OctoSketch Integration and Enhancements (November–December). Sketch-based data structures were integrated to enable scalable per-IP traffic monitoring. The architecture was extended to support the DPI-Sketch and Sketch-ADV feature extraction configurations, enabling feature derivation from packet headers alone.

Block E – Data Capture (December). Systematic data collection was performed in the CloudLab testbed. Multiple attack types were replayed under controlled conditions

following the baseline attack recovery protocol, enabling the generation of labelled multi-class datasets for supervised learning under both feature configurations.

Block F – ML Integration (January). Supervised ML models were trained on the collected datasets for both DPI-Sketch and Sketch-ADV configurations. The trained models were exported and embedded directly into the DPDK-based detector through the native C inference interface, enabling real-time classification at the level of detection windows.

Block G – Improvements and Documentation (February–March). The final phase covers refinement of feature extraction strategies, comparative evaluation of DPI-Sketch versus Sketch-ADV classification performance, system performance analysis, and preparation of the final thesis documentation.

Chapter 5

System Development

5.1 Testbed Infrastructure

All experiments are conducted on a controlled four-node deployment hosted on CloudLab [16]. The testbed is designed to reproduce realistic high-speed network conditions in a reproducible and isolated environment, enabling controlled generation of both benign and attack traffic under known ground-truth labels.

All four nodes are homogeneous instances of the CloudLab `c6525-25g` hardware profile (Dell PowerEdge C6525), with the following specifications:

- **CPU:** AMD EPYC 7302P, 16 cores at 3.0 GHz
- **Memory:** 128 GB DDR4 ECC
- **NIC:** Dual-port Mellanox ConnectX-5, 25 Gbps per port
- **OS:** Ubuntu 20.04 LTS
- **Hugepages (monitor node only):** 512 pages of 2 MB reserved at boot for DPDK

Figure 5.1 illustrates the topology. The four nodes are interconnected through 25 Gbps links, and each node is assigned a distinct IP subnet that serves as the ground-truth label source for traffic classification.

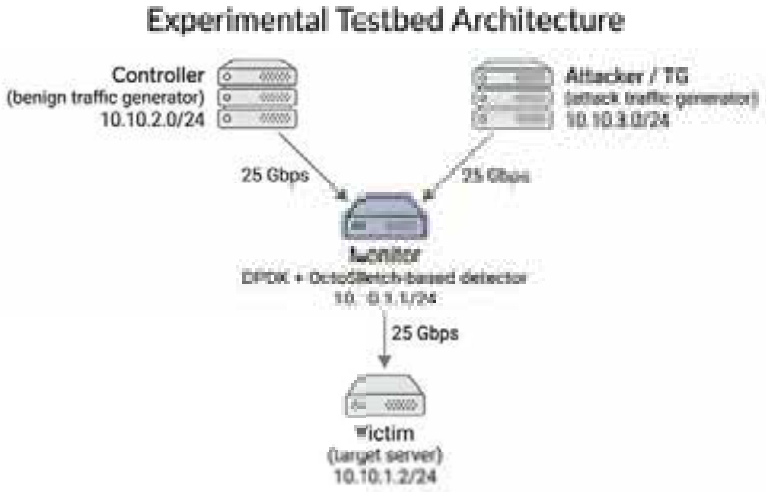

Figure 5.1: CloudLab testbed topology.

5.1.1 Node Roles

- **Monitor (10.10.1.1).** The detection node. It runs the DPDK-based detector in promiscuous mode on its 25 Gbps NIC, processing all traffic forwarded through the link. This node dedicates its CPU cores exclusively to packet processing and ML inference.
- **Victim (10.10.1.2).** The target server. It receives all traffic directed at the monitored subnet and acts as the destination for both benign and attack flows.
- **Controller (10.10.2.x).** The benign traffic generator. It produces realistic background traffic towards the victim at configurable rates, simulating legitimate user activity during experiments.
- **Attacker (10.10.3.x).** The attack traffic generator. It replays DDoS attack scenarios at controlled rates and durations, covering the attack categories present in the evaluation dataset.

5.1.2 Network Topology and IP-Based Labelling

The IP addressing scheme is central to the data labelling strategy. Because all attack traffic originates from the 10.10.3.x subnet and all benign traffic from the 10.10.2.x subnet, each captured detection window can be automatically labelled without manual annotation. The monitor node identifies the traffic composition of each 50 ms window by tracking per-source-subnet packet counts, enabling the construction of accurately labelled multi-class datasets for model training and evaluation. The source subnet is used exclusively for labelling purposes during dataset construction and is not exposed

to the classifier as a feature; the model learns from statistical traffic patterns that generalise across arbitrary IP ranges.

Each experiment follows a structured baseline–attack–recovery protocol: a period of benign-only traffic precedes the attack injection, followed by a recovery phase after the attack generator is stopped. This protocol ensures that the dataset contains representative samples of both benign and attack conditions, and that transient periods at attack onset and cessation are correctly identified during labelling.

5.1.3 Traffic Generators

Traffic generation follows a two-step offline process: a Python-based generator produces a PCAP file (a standard binary format that stores captured network packets along with their timestamps and metadata), which is then replayed at line rate by a DPDK-based sender. This separation allows precise control over traffic composition without the constraint of real-time packet crafting.

Attack Generator

The attack generator is implemented in Python using the Scapy packet crafting library [30]. It produces DDoS traffic following the attack taxonomy of the CIC-DDoS-2019 dataset [21], supporting 13 individual attack types. Each type constructs packets that replicate the header and payload structure of the corresponding real-world attack. Table 5.1 summarises the transport-layer properties of each type.

Packet construction is performed at the Scapy layer stack level. As an example, a SYN flood packet is built as:

```
pkt = (IP(src=attacker_ip, dst=target_ip) /
       TCP(sport=random.randint(1024, 65535),
           dport=random.choice([80, 443, 22, 23]),
           flags='S',
           seq=random.randint(1000, 4000000000),
           window=random.choice([1024, 2048, 4096, 8192])))
```

Listing 5.1: SYN flood packet construction.

Amplification attacks follow the same pattern, crafting valid protocol requests (e.g., DNS ANY queries, NTP MONLIST, SNMP GetBulk) that would elicit large responses from real servers. Source addresses are drawn from a pool of configurable size within the 10.10.3.x botnet subnet, with each packet selecting a source IP at random to simulate a distributed botnet.

In mixed mode, the user specifies a set of attack types and their percentage contributions. The generator builds a weighted distribution list and selects the attack type for each packet by random sampling, naturally enforcing the desired proportions across the output PCAP:

```
# Build weighted distribution: e.g., 60% SYN + 40% UDP
phase_distribution = (['syn'] * 60) + (['udp'] * 40)

selected_attack = random.choice(phase_distribution)
```

Listing 5.2: Mixed mode packet type selection.

Table 5.1: Attack types supported by the traffic generator, with their transport-layer characteristics.

Attack type	Protocol	Dst port	Packet characteristics
SYN flood	TCP	80/443/22,...	TCP SYN, no payload, random source port
UDP flood	UDP	random	Random payload, 64–512 bytes
UDP large	UDP	random	Random payload, 1000–1400 bytes
WebDDoS	TCP	80/443	SYN + HTTP GET/POST request
DNS amp.	UDP	53	DNS ANY query (real domain names)
NTP amp.	UDP	123	NTP MONLIST request
SNMP amp.	UDP	161	SNMP GetBulk request
SSDP amp.	UDP	1900	M-SEARCH request
PortMap amp.	UDP	111	RPC GETPORT call
NetBIOS amp.	UDP	137/138	NBNS name query or datagram
LDAP amp.	UDP	389	LDAP search request
MSSQL amp.	UDP	1434	MSSQL ping request
TFTP amp.	UDP	69	TFTP read request

Benign Generator

The benign generator is also implemented in Python with Scapy and produces realistic background traffic composed of five protocol classes: HTTP (port 80), DNS (port 53), SSH (port 22), ICMP echo, and background UDP. Each protocol class generates complete, multi-packet flows rather than isolated packets. HTTP flows include a full TCP three-way handshake followed by a GET request and a response; SSH flows simulate a sequence of encrypted data exchanges over an established connection; DNS flows consist of a query and a response with one to four answer records.

To capture the temporal variability of real network activity, traffic is structured into four sequential phases, each with a distinct protocol mix, packet rate multiplier, and

inter-packet jitter:

Table 5.2: Benign traffic generation phases and protocol composition.

Phase	Name	Protocol mix	Intensity	Jitter
1	HTTP peak	70% HTTP, 15% DNS, 5% SSH, 5% ICMP, 5% UDP	1.3×	20 ms
2	DNS burst	30% HTTP, 50% DNS, 5% SSH, 10% ICMP, 5% UDP	0.8×	50 ms
3	SSH stable	35% HTTP, 10% DNS, 40% SSH, 5% ICMP, 10% UDP	0.6×	10 ms
4	UDP light	25% HTTP, 15% DNS, 10% SSH, 15% ICMP, 35% UDP	0.5×	80 ms

The core generation loop iterates over phases and selects the protocol class for each flow by sampling from a weighted distribution, mirroring the same mechanism used in the attack generator:

```
for phase in TRAFFIC_PHASES:
    phase_distribution = phase.get_traffic_distribution()
    while packets_in_phase < phase_target:
        traffic_type = random.choice(phase_distribution)
        if traffic_type == 'http':
            flow_packets = generate_http_traffic(client_ip,
              server_ip, ...)
        elif traffic_type == 'dns':
            flow_packets = generate_dns_query(client_ip,
              server_ip, ...)
        # ... ssh, icmp, udp
```

Listing 5.3: Benign traffic phase loop.

Source addresses are drawn from a configurable pool of up to 500 client IPs within the 10.10.2.x subnet. A parallel variant distributes generation across multiple CPU cores to accelerate the production of large-scale datasets.

5.1.4 Traffic Senders

Both the attack and benign senders are built on the same DPDK-based PCAP replay engine. They share the same codebase and support the same three replay modes; the only difference is the packet classification logic used in adaptive mode, which categorises

packets by attack type in the attack sender and by application protocol in the benign sender.

- **Fast mode** (default): transmits packets at maximum NIC rate, ignoring PCAP timestamps. Used for bulk data collection where temporal fidelity is less important than throughput. The core loop allocates a burst of mbufs, copies PCAP packet data into each one, and dispatches the entire burst in a single DPDK call:

```
rte_pktmbuf_alloc_bulk(mbuf_pool, pkts, BURST_SIZE);
for (i = 0; i < BURST_SIZE; i++) {
    char *buf = rte_pktmbuf_mtod(pkts[i], char *);
    rte_memcpy(buf, pcap_packets[idx].data, pcap_packets[
        idx].len);
    pkts[i]->data_len = pkts[i]->pkt_len = pcap_packets[idx
        ].len;
    if (++idx >= num_pcap_packets) idx = 0;  /* loop PCAP
        */
}
nb_tx = rte_eth_tx_burst(port_id, 0, pkts, BURST_SIZE);
```

Listing 5.4: Fast mode burst transmission loop.

- **Timed mode** (`--pcap-timed`): replays packets respecting the original PCAP timestamps. A `--speedup` multiplier compresses or expands the playback rate relative to real time. For each packet, the inter-packet delay is read from the PCAP timestamps and divided by the speedup factor before blocking:

```
uint64_t delta_us = timeval_diff_us(&prev_ts, &pkt->
    timestamp);
delta_us = delta_us / replay_cfg.speedup_factor;
if (delta_us > 0 && delta_us < 10000000)
    rte_delay_us_block(delta_us);  /* busy-wait,
        microsecond precision */
prev_ts = pkt->timestamp;
```

Listing 5.5: Timed mode inter-packet delay.

- **Adaptive mode** (`--adaptive-attack` / `--adaptive`): uses the PCAP as a random packet pool and generates a continuous traffic stream according to a user-defined phase schedule in JSON format. Each phase specifies its duration and the fractional composition of traffic types. Enables long duration experiments

with configurable attack intensity profiles without being limited by the size of the pre-generated PCAP.

Both senders accept a target line rate via `--rate-gbps` and a duration limit via `--duration`, and support looping for extended experiments. Two input modes are available:

- **Single file**: a single PCAP file is loaded and replayed.
- **Directory**: a directory of multiple PCAP files is loaded and replayed in sequence.

5.2 Detector Architecture

5.2.1 DPDK Integration

The detector is built on top of DPDK's Environment Abstraction Layer (EAL), which is the first component initialised at start-up via `rte_eal_init()`. The EAL handles hugepage memory reservation, CPU core affinity binding, and poll-mode driver (PMD) setup for the NIC. A single packet buffer pool of 524,288 mbufs with a per-core cache of 512 entries is then allocated on the local NUMA node via `rte_pktmbuf_pool_create()`, ensuring that packet buffers are physically close to the cores that will process them and avoiding expensive cross-socket memory accesses.

- **Port and RSS configuration.** The NIC is configured with 14 RX queues and 1 TX queue via `rte_eth_dev_configure()`, with a receive descriptor ring of 32,768 entries and a transmit ring of 4,096 entries, sized to absorb burst traffic without overflow. RSS is enabled with a hash over IP addresses and TCP/UDP port fields (`ETH_RSS_IP | ETH_RSS_TCP | ETH_RSS_UDP`) using the default Toeplitz key, ensuring all packets of a given flow are consistently directed to the same queue and the same worker core. Promiscuous mode is enabled via `rte_eth_promiscuous_enable()`, required for passive monitoring since the detector does not own any IP addresses in the inspected traffic.

```
struct rte_eth_conf port_conf = {
    .rxmode = { .mq_mode = ETH_MQ_RX_RSS },
    .rx_adv_conf = {
        .rss_conf = {
            .rss_key = NULL, /* use default Toeplitz key */
            .rss_hf  = ETH_RSS_IP | ETH_RSS_TCP | ETH_RSS_UDP,
        },
    },
};
```

```
rte_eth_dev_configure(port, 14 /* rx_queues */, 1 /* tx_queues */,
                      &port_conf);
rte_eth_promiscuous_enable(port);
```

Listing 5.6: NIC port configuration: RSS over IP and transport ports, 14 RX queues.

- **Worker polling loop.** Each worker runs a tight `while(!force_quit)` loop with no system calls or interrupts. `rte_eth_rx_burst()` retrieves up to 2,048 packets per call directly from the NIC descriptor ring. To keep the CPU pipeline full, the first 16 packet pointers of each burst are prefetched into L1 cache before the processing loop begins (pre-loop prefetch), and an additional `rte_prefetch0()` 16 positions ahead is issued inside the loop (rolling prefetch). Once processed, each mbuf is returned to the pool via `rte_pktmbuf_free()`, completing the zero-copy cycle with no heap allocation at any point.

```
while (!force_quit) {
    struct rte_mbuf *bufs[BURST_SIZE]; /* BURST_SIZE = 2048 */
    uint16_t nb_rx = rte_eth_rx_burst(port, queue_id, bufs,
        BURST_SIZE);
    if (unlikely(nb_rx == 0)) continue;

    /* Prefetch first 16 packets before processing loop */
    for (uint16_t i = 0; i < nb_rx && i < 16; i++)
        rte_prefetch0(rte_pktmbuf_mtod(bufs[i], void *));

    for (uint16_t i = 0; i < nb_rx; i++) {
        if (i + 16 < nb_rx)                          /* rolling
            prefetch */
            rte_prefetch0(rte_pktmbuf_mtod(bufs[i + 16], void
                *));
        process_packet(bufs[i]);                      /* parse +
            update */
        rte_pktmbuf_free(bufs[i]);                    /* return to
            pool */
    }
}
```

Listing 5.7: Worker inner loop: burst receive, prefetch pipeline, and mbuf recycling.

5.2.2 Multi-Core Design

The system follows a strict producer–consumer separation between worker cores and the coordinator core. Fourteen worker threads, pinned to lcores 1–14, each handle one RSS queue exclusively: they parse packet headers, update per-worker protocol counters, and insert packet records into their local OctoSketch instance. Because each worker operates entirely on thread-local data structures, no locks or atomic operations are required on the packet processing hot path.

A single coordinator thread runs on lcore 15 and is responsible for all cross-worker aggregation and detection logic. Every 50 ms, the coordinator reads the per-worker protocol counters, merges the 14 per-worker sketch instances into a global merged sketch, and extracts the feature vector for the current detection window. This architecture decouples the high-frequency packet processing path from the lower-frequency detection logic, preventing any contention between the two.

- **Thread launch.** All threads are started via `rte_eal_remote_launch()`: the 14 worker threads are bound to lcores 1–14, each receiving its queue identifier as argument, and the coordinator thread is bound to lcore 15. The main thread calls `rte_eal_mp_wait_lcore()` and blocks until a `SIGINT` sets the `force_quit` flag.

- **Window timing.** The coordinator measures elapsed time using the CPU Time Stamp Counter (TSC) via `rte_get_tsc_cycles()` and `rte_get_tsc_hz()`, with no blocking sleep. At each iteration it checks whether $(t_{\text{now}} - t_{\text{last}})/f_{\text{TSC}} \geq 50\,\text{ms}$ (`FAST_DETECTION_INTERVAL = 0.05`); only then does it proceed with sketch merging and feature extraction, guaranteeing the coordinator core is never idle-blocked.

- **Worker-to-coordinator communication.** Each worker writes its cumulative counters into a dedicated entry of the global `g_worker_stats[]` array without synchronisation. At each window boundary the coordinator sums all 14 entries to obtain cumulative totals, then subtracts the snapshot from the previous boundary to isolate the per-window delta. This lock-free approach is acceptable because the statistical nature of the features tolerates the small races that may occur during the brief read window.

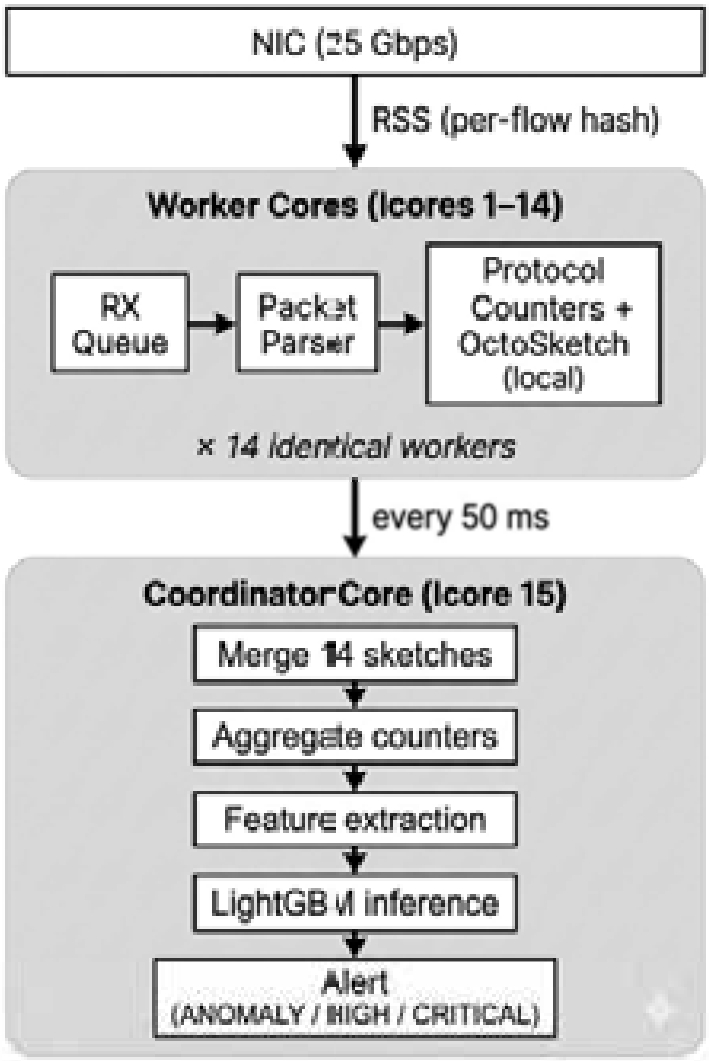

Figure 5.2: Multi-core detector architecture: 14 worker threads.

5.2.3 OctoSketch Integration

Each worker maintains a local OctoSketch instance configured with 8 hash rows and 4 096 buckets per row. The counter matrix itself occupies approximately 128 KB, but each sketch instance also includes per-bucket IP tracking structures for heavy-hitter detection, bringing the total memory footprint to approximately 640 KB per instance, consistent with the figure reported in Chapter 2. To bound the overhead imposed on the packet processing path, sketch updates are applied with a sampling rate of 1 in every 32 packets, reducing the per-packet cost while preserving statistical accuracy for high-volume flows.

The coordinator performs a full merge of all 14 worker sketches every detection window. The merge accumulates bucket counts by summation across all workers and retains the heaviest-contributing IP address per bucket, enabling top-K heavy-hitter extraction on the merged result. In addition to the 50 ms operational sketch, the system maintains

separate sketch instances aggregated over 1 s, 10 s, and 1 minute, providing multi-scale traffic summaries used as temporal features in the ML feature vector. Each worker tracks a window counter per time scale and resets the corresponding sketch once the counter reaches its threshold: every 20 windows for the 1 s sketch, every 200 windows for the 10 s sketch, and every 1,200 windows for the 1 min sketch (`SCALE_1S_WINDOWS`, `SCALE_10S_WINDOWS`, `SCALE_1MIN_WINDOWS`).

In Sketch ADV mode, 12 additional per-protocol sketches are maintained in parallel, one for each of the monitored attack protocols: DNS, NTP, SNMP, SSDP, PortMap, NetBIOS, LDAP, MSSQL, TFTP, TCP SYN, HTTP, and residual UDP. These protocol sketches are updated during packet classification on each worker and merged by the coordinator alongside the global sketch.

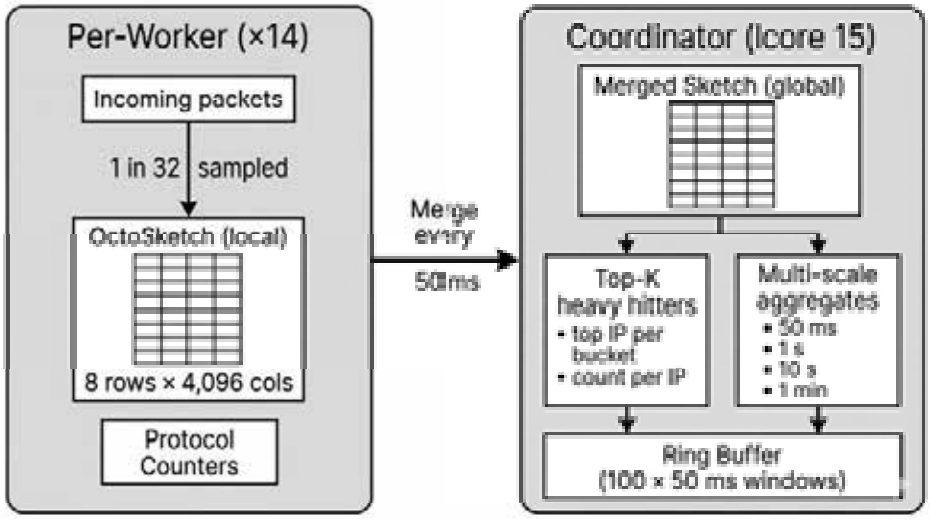

Figure 5.3: OctoSketch integration.

After each merge, the coordinator appends the aggregated statistics of the current window to a circular **ring buffer** of 100 entries, representing the last 5 seconds of traffic history (100 windows × 50 ms). The ring buffer acts as the bridge between the sketch layer and the ML feature extraction layer: rather than classifying each window in isolation, the feature extractor reads back into the buffer to compute temporal metrics such as rate-of-change over the last 5 and 10 windows, traffic variance over the last 20 windows, a running baseline average, and an adaptive detection threshold derived from the 3-sigma rule applied to the recent history. These temporal features allow the model to distinguish sustained attack patterns from transient traffic spikes that would be indistinguishable from a single instantaneous snapshot.

5.3 Feature Extraction System

The feature extraction layer is the bridge between the raw packet statistics accumulated by the detector and the ML classifier. Every 50 ms, once the coordinator has merged all worker sketches and aggregated protocol counters, a structured feature vector is assembled from the available data and passed to the embedded LightGBM model. The system implements two distinct extraction modes whose design reflects the research comparison at the core of Objective 4 (Section 4.2).

DPI-Sketch as accuracy reference. DPI-Sketch extends the global OctoSketch layer with a DPI parser that inspects the application-layer content of each packet. The resulting 75-feature vector combines header-derived sketch statistics with payload-derived protocol counters, and represents the *accuracy ceiling*: the classification performance achievable when payload access is unrestricted and both sources of evidence are available simultaneously. It is used in this work as the reference against which the sketch-only contribution is measured.

A conventional DPI system from the literature would be an unsuitable baseline for this comparison. Such a system would employ a fundamentally different architecture — per-flow connection tables, stream reassembly buffers, session state machines, rather than the OctoSketch infrastructure developed here. Using it as a reference would conflate two independent variables: the architectural contribution of the sketch layer and the informational contribution of payload features. It would then be impossible to isolate the accuracy cost of removing payload inspection from the broader differences in system design.

DPI-Sketch solves this by sharing the *identical* OctoSketch architecture as Sketch-ADV: both modes run the same DPDK pipeline, the same worker and coordinator structure, and the same sketch data structures. The only variable that differs between the two modes is whether the DPI parser is active during packet processing. The accuracy gap between the two modes therefore reflects exclusively the discriminative value of payload-derived features, free from any architectural confound.

Sketch-ADV as the proposed deployable mode. Sketch-ADV derives all 64 features exclusively from packet headers, extending the sketch layer with 12 per-protocol OctoSketch instances in place of the DPI parser. It represents the primary contribution of this work: a header-only detector that, built on the same sketch foundation, aims to match the accuracy of the DPI-enhanced reference while remaining applicable to encrypted traffic.

The following subsections describe the implementation of each mode in detail.

5.3.1 DPI-Sketch Mode

DPI-Sketch is the payload-aware configuration. It constructs a 75-feature vector by combining features derived from deep packet inspection with temporal statistics from the global OctoSketch ring buffer. The feature vector is assembled in the coordinator thread immediately after each 50 ms detection window closes.

The 75 features are organised into six functional groups, summarised in Table 5.3:

Table 5.3: DPI-Sketch feature groups and their composition (75 features total).

Group	Features	Description
Raw counters	10	Total packets, bytes, UDP, TCP, ICMP, SYN, HTTP, DNS, baseline and attack packet counts
Basic ratios	4	UDP/TCP ratio, SYN percentage, baseline/attack ratio, bytes per packet
Protocol-specific	22	Per-protocol payload fields: DNS query types, NTP mode bytes, SNMP OIDs, HTTP request counts, amplification response sizes
Amplification ratios	6	Response-to-request size ratios and amplification factors per protocol
Normalised fractions	15	Per-protocol share of total traffic, protocol diversity index
Temporal (sketch)	14	Ring buffer features: rate-of-change over 5 and 10 windows, variance over 20 windows, running baseline, adaptive threshold

The feature vector is assembled sequentially using an index `fi` that increments through the six groups in order. The following excerpt shows the structure of the assembly:

```
int fi = 0;
/* Group 1 - Raw counters (10) */
features[fi++] = (double)ml_total_packets;
features[fi++] = (double)ml_total_bytes;
features[fi++] = (double)ml_udp_packets; /* ... */

/* Group 2 - Basic ratios (4) */
features[fi++] = (ml_tcp_packets > 0)
    ? (double)ml_udp_packets / ml_tcp_packets : 0.0;
features[fi++] = (ml_total_packets > 0)
    ? (double)ml_syn_packets / ml_total_packets : 0.0; /* ...
```

```
12
13 /* Group 3 - Protocol-specific counters (22) */
14 features[fi++] = (double)ml_ntp_monlist;
15 features[fi++] = avg_ntp_resp;  /* avg NTP response size */
16 features[fi++] = (double)ml_dns_any;
17 features[fi++] = avg_dns_resp;  /* avg DNS response size */ /*
       ... */
18
19 /* Group 4 - Amplification ratios (6) */
20 features[fi++] = (ml_ntp_monlist > 0) ? avg_ntp_resp / 48.0 :
       0.0;
21 features[fi++] = (ml_dns_any > 0)     ? avg_dns_resp / 60.0 :
       0.0; /* ... */
22
23 /* Group 5 - Normalised protocol fractions (15) */
24 features[fi++] = proto_ratios[0];  /* syn_only / total */
25 features[fi++] = proto_ratios[2];  /* dns_queries / total */ /*
       ... */
26
27 /* Group 6 - Temporal sketch features (14) from ring buffer */
28 features[fi++] = (double)fw.delta_pps_5w;
29 features[fi++] = (double)fw.delta_pps_10w;
30 features[fi++] = (double)fw.pps_variance;
31 features[fi++] = (double)fw.pps_baseline;  /* fi == 75 */
```

Listing 5.8: DPI-Sketch feature vector assembly structure (75 features).

The protocol-specific and amplification features are extracted by inspecting the payload of each packet during the per-worker processing step. For example, a DNS packet is classified as an amplification attempt if its query type field equals ANY (type 255), and its response size is recorded to compute the amplification ratio. The corresponding feature construction in C reads:

```
1 if (dst_port == 53 || src_port == 53) {
2     stats->dns_packets++;
3     if (payload_len > 0) {
4         uint8_t qtype = payload[payload_len - 1];  /*
              simplified */
5         if (qtype == 255) stats->dns_any_queries++;
6         stats->dns_response_size_total += payload_len;
7     }
```

```
}
/* Amplification ratio computed per window */
features[F_DNS_AMP_RATIO] = (float)stats->
    dns_response_size_total
                              / (stats->dns_packets + 1);
```

Listing 5.9: DPI payload inspection for DNS amplification features.

The 14 temporal features are appended last and are identical across both modes. They are derived from the ring buffer described in Section 5.2.3, providing the model with a short-term history of traffic behaviour rather than a purely instantaneous snapshot.

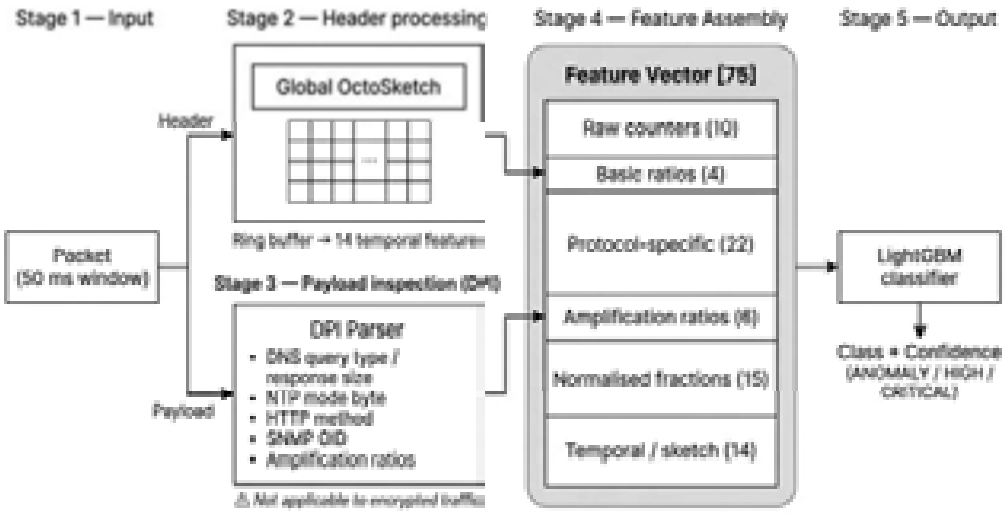

Figure 5.4: DPI-Sketch feature extraction pipeline (75 features).

Figure 5.4 illustrates the complete DPI-Sketch pipeline. Each incoming packet is split into two parallel processing paths:

- **Header path:** feeds the global OctoSketch instance maintained per worker. Source IP, destination port, protocol, and packet size are used to update the sketch counters. At the end of each 50 ms window, the coordinator merges all worker sketches and queries the ring buffer to produce the 14 temporal features.
- **Payload path:** runs the DPI parser, which inspects the application-layer content of each packet to extract protocol-specific fields: DNS query type and response size, NTP request mode byte, HTTP method and URI, SNMP OID, and raw payload length for amplification ratio computation.

Both paths feed their outputs into the feature assembly stage, where the six feature

groups are concatenated into a single 75-dimensional vector. This vector is then normalised using the pre-fitted StandardScaler and passed to the embedded LightGBM classifier, which outputs a class label and a confidence score used to determine the alert level.

DPI-Sketch presents one structural limitation with respect to Sketch-ADV: it cannot be applied to encrypted traffic. Once the transport payload is opaque, fields such as DNS query type, NTP mode byte, or HTTP method are inaccessible, and the 28 payload-derived features become unavailable. Unlike traditional DPI systems that maintain per-flow connection tables and reassembly buffers, DPI-Sketch in this implementation computes all payload-derived features as aggregate window counters, running totals per protocol per 50 ms detection window, so its memory footprint remains bounded and independent of flow cardinality. Sketch-ADV addresses the encryption limitation by replacing the DPI parser with 12 additional per-protocol sketch instances, deriving all features exclusively from packet headers, as described in the following section.

A complete description of all 75 DPI-Sketch features, including the group they belong to, how each value is obtained, and whether it is an exact counter or an estimate, is provided in Appendix B.

5.3.2 Sketch-ADV Mode

Sketch-ADV is the header-only configuration. It constructs a 64-feature vector exclusively from packet header fields, without any payload access. Instead of relying on a DPI parser, it extends the global OctoSketch with 12 additional per-protocol sketch instances, one for each of the monitored attack protocol categories. Each worker updates the corresponding protocol sketch on every sampled packet based solely on the destination port and transport protocol fields of the IP header.

The 64 features are organised into three groups, summarised in Table 5.4:

Table 5.4: Sketch-ADV feature groups and their composition (64 features total).

Group	Features	Description
Global sketch (temporal)	14	Rate-of-change over 5 and 10 windows, variance, baseline, top-IP pps at 50 ms/1 s/1 min, heavy-hitter count, IP concentration, new-IP ratio, entropy, adaptive threshold
Per-protocol sketches	48	For each of the 12 protocols (DNS, NTP, SNMP, SSDP, PortMap, NetBIOS, LDAP, MSSQL, TFTP, SYN, HTTP, UDP-other): packets per second, heavy-hitter count, top-IP concentration, protocol share of total traffic
Packet size	2	Mean packet size and variance over the window

The 12 per-protocol sketches are queried in a uniform loop: for each protocol, the coordinator retrieves the total packet count, queries the top-K heavy hitters, and computes the four features from the sketch statistics alone, with no reference to packet payloads:

```
struct octosketch *proto_merged[12] = {
    &g_merged_sketch_dns, &g_merged_sketch_ntp,  &
       g_merged_sketch_snmp,
    &g_merged_sketch_ssdp,&g_merged_sketch_portmap,&
       g_merged_sketch_netbios,
    &g_merged_sketch_ldap,&g_merged_sketch_mssql, &
       g_merged_sketch_tftp,
    &g_merged_sketch_syn, &g_merged_sketch_http,  &
       g_merged_sketch_udp_other,
};
uint64_t total_global = octosketch_get_total(&
   g_merged_sketch_attack);

for (int p = 0; p < 12; p++) {
    uint64_t total_proto = octosketch_get_total(proto_merged[p
       ]);
    struct heavy_hitter top5[5];
    octosketch_top_k(proto_merged[p], 5, top5);

    features[fi++] = (double)total_proto / window_sec;        /*
        pps */
```

```
    features[fi++] = (double)ccunt_heavy_hitters(top5, 5);    /*
        hnavy hitters */
    features[fi++] = (double)tcp5[0].count / (total_proto+1);/*
        IP concentration */
    features[fi++] = (double)tctal_proto / (total_global+1); /*
        ratio vs total */
}
```

Listing 5.10: Sketch-ADV per-protocol feature extraction loop (12 protocols × 4 features).

The elegance of this design is that all 12 protocols are handled by the same four lines of code, iterating over an array of sketch pointers. Adding a new protocol category requires only inserting a new sketch instance into the array, with no changes to the feature extraction logic.

The packet size features are derived from the mean and variance of packet lengths sampled during the window, computed from per-worker accumulators using the computational formula $\mathrm{Var}(x) = \mathbb{E}[x^2] - \mathbb{E}[x]^2$, which avoids storing individual packet sizes.

Memory efficiency. The primary advantage of Sketch-ADV over traditional DPI-based approaches is its bounded memory footprint. Every per-protocol sketch is a fixed structure of 8 rows × 4 096 columns regardless of traffic volume or flow cardinality. The complete set of merged sketches at the coordinator (1 global + 12 per-protocol) occupies a constant 13×640 KB ≈ 8 MB, independent of the number of active flows. By contrast, DPI-based approaches that maintain per-flow connection state can require several gigabytes under high-cardinality traffic conditions.

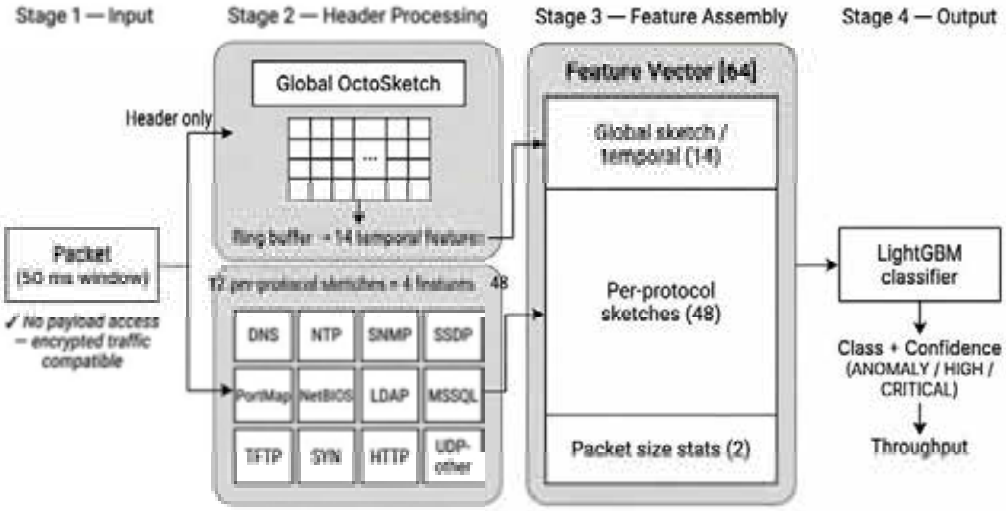

Figure 5.5: Sketch-ADV feature extraction pipeline (64 features, header-only).

Figure 5.5 illustrates the Sketch-ADV pipeline. Unlike DPI-Sketch, all information flows through a single **header path**: each packet updates both the global OctoSketch and the corresponding per-protocol sketch based solely on its destination port and protocol field. No payload is accessed at any point. At the end of each 50 ms window, the coordinator merges all worker sketches and extracts the three feature groups: the 14 temporal features from the global sketch ring buffer, the 48 per-protocol features from the 12 protocol sketches, and the 2 packet size statistics. The resulting 64-dimensional vector is normalised and passed to the same embedded LightGBM classifier used in DPI-Sketch mode.

A fundamental difference between the two modes lies in the nature of the features themselves. DPI-Sketch constructs its feature vector from both direct payload counters, such as `ml_dns_any` or `ml_ntp_monlist`, which are exact integer counts incremented on every packet, and sketch-derived estimates for header-level aggregates such as heavy-hitter IP rates and temporal ratios. Sketch-ADV relies entirely on probabilistic estimates derived by querying the per-protocol sketch instances; all values carry a bounded approximation error inherent to the Count-Min Sketch, controlled by the sketch dimensions (8 rows × 4 096 columns). Both modes share the same fixed-memory OctoSketch infrastructure and represent a significant improvement over traditional DPI-based flow approaches, which require per-flow hash tables that grow unboundedly with traffic cardinality and introduce detection latencies of tens to hundreds of seconds. Within this shared foundation, the core architectural distinction is not memory cost but feature composition: DPI-Sketch trades encryption compatibility for a richer, partially exact feature set, while Sketch-ADV achieves full encryption transparency at the cost of operating exclusively on probabilistic header-level aggregates. Evaluating the impact of

this trade-off on classification accuracy is one of the primary objectives of this work.

A complete description of all 64 Sketch-ADV features, including the group they belong to, how each value is obtained, and whether it is an exact counter or a sketch-derived estimate, is provided in Appendix B.

5.3.3 Embedded Inference

Once the feature vector has been assembled (75 features in DPI-Sketch mode or 64 in Sketch-ADV mode), the coordinator thread passes it to the embedded inference module. The inference runs entirely in-process using the LightGBM C API [15], with no external processes, no inter-process communication, and no network calls. The model, the feature scaler, and the class label mapping are all pre-trained offline and loaded into memory at detector startup; the runtime system is read-only with respect to the model.

Model Loading

At initialisation, `ml_init()` loads three artefacts from a configurable directory:

- `lightgbm_model.txt`: the serialised LightGBM booster, loaded via `LGBM_BoosterCreateFromModelfile()`. The number of features and output classes is queried immediately after loading to verify consistency with the compiled detector.
- `feature_scaler.json`: a JSON file containing two arrays, `mean` and `scale`, with one entry per feature. These are the per-feature mean and standard deviation computed during training using scikit-learn's `StandardScaler`.
- `label_mapping.json`: a JSON mapping from class index to class name (e.g. `0` → `benign`, `1` → `dns_amp`, `2` → `ntp_amp`, and so on for each attack category).

```
/* 1. Load serialised LightGBM model */
LGBM_BoosterCreateFromModelfile(model_path, &num_iterations,
                                &model->booster);
LGBM_BoosterGetNumFeature(model->booster, &model->num_features)
    ;
LGBM_BoosterGetNumClasses(model->booster, &model->num_classes);

/* 2. Load StandardScaler (mean + scale arrays from JSON) */
parse_json_array(scaler_json, "mean",  model->scaler_mean,  N);
parse_json_array(scaler_json, "scale", model->scaler_scale, N);

/* 3. Load label mapping (class index -> name) */
```

```
parse_json_string(mapping_json, "", i, model->class_names[i],
    ...);
```

Listing 5.11: Model initialisation via LightGBM C API (`ml_init()`).

Per-Window Inference

At the end of each 50 ms window, if the model is loaded and the window contains more than 100 packets, the coordinator calls `ml_predict()`. This function performs two sequential steps:

1. **Feature normalisation.** Each feature is standardised using the pre-fitted scaler: $x'_i = (x_i - \mu_i)/\sigma_i$. This step is critical because LightGBM was trained on normalised features; skipping it would produce meaningless predictions.

2. **Classification.** `LGBM_BoosterPredictForMat()` runs a single-row prediction and returns a probability vector with one entry per class. The predicted class is the argmax of this vector, and the confidence is the corresponding probability value.

```
int ret = ml_predict(g_ml_model, features, num_features, &
    ml_pred);

if (ret == 0 && ml_pred.predicted_class != 0) { /* not benign
    */
    float conf = ml_pred.confidence;
    if      (conf >= 0.75f) alert_level = ALERT_CRITICAL;
    else if (conf >= 0.50f) alert_level = ALERT_HIGH;
    else                    alert_level = ALERT_LOW; /* ANOMALY
        */
}
```

Listing 5.12: Per-window inference and alert level assignment in the coordinator thread.

Alert Levels

The alert level is determined solely by the confidence score of the predicted attack class. If the model predicts class 0 (benign), no alert is raised regardless of the confidence value. Table 5.5 summarises the three alert levels.

Table 5.5: Alert levels based on ML confidence score.

Level	**Confidence**	**Meaning**
ANOMALY	$< 50\%$	Attack class predicted with low confidence. Possible anomaly or borderline traffic pattern worth monitoring.
HIGH	$50\% \leq c < 75\%$	Moderately confident attack prediction.
CRITICAL	$\geq 75\%$	High-confidence attack classification.

The alert message includes the predicted class name and the confidence percentage, allowing operators to distinguish between a definitive detection and a borderline case. The minimum 100-packet threshold prevents the model from running on near-empty windows where the sketch statistics carry no meaningful signal.

Chapter 6

Experimental Evaluation

6.1 Traffic Generation Setup

All experiments were conducted on the four-node CloudLab testbed described in Section 5.1. The controller node replays benign background traffic towards the victim, the attacker node replays DDoS attack PCAPs, and the monitor node runs the DPDK-based detector and records all output logs. The following subsections describe how each traffic source was constructed and the timing protocol that governs each experimental run.

6.1.1 Attack Traffic

Individual Attacks

Attack traffic was generated synthetically using a purpose-built Python generator modelled after the thirteen attack classes defined in CIC-DDoS-2019 [21]. The classes are organised into three categories, each with a distinct mechanism and network-level signature:

- **Reflection/amplification attacks** (NTP, DNS, SNMP, SSDP, Portmap, NetBIOS, LDAP, MSSQL, TFTP). The attacker sends small requests to publicly reachable servers using a spoofed source address set to the victim's IP. The servers reply to the victim with responses that are orders of magnitude larger than the original request. The bandwidth delivered to the victim is therefore determined by the amplification factor of the exploited protocol rather than by the attacker's uplink capacity, making this category disproportionately destructive at low injection rates.
- **Volumetric and connection-exhaustion floods** (UDP flood, SYN flood).

UDP flood saturates the victim's link by injecting a continuous stream of large UDP datagrams, consuming raw bandwidth. SYN flood instead targets the victim's TCP connection state: each forged SYN causes the kernel to allocate a half-open connection entry, and a sustained high-rate stream exhausts the SYN backlog before any legitimate connection can complete.

- **Application-layer attacks** (WebDDoS). Full TCP connections are established with the victim's HTTP server, followed by well-formed GET or POST requests. Because each flow completes the three-way handshake, stateless packet filters cannot block it, and the attack pressure is exerted on the web server's request-processing capacity rather than on the link.

One PCAP file was produced for each individual attack class, plus four additional mixed-class PCAPs. All PCAPs were generated with 200 simulated attacker IPs drawn from the `10.10.3.0/24` prefix and a fixed victim at `10.10.1.2`, using 16 parallel worker processes.

The generation parameters were tuned per attack class to reflect realistic traffic profiles (Table 6.1):

- **High-rate single-packet types** (SYN, UDP, and NTP injection requests) use the highest packet counts (20 M) and intensity factors (3.0), as each injected unit is a single small packet and large volumes are needed to produce an observable signature at the monitor. For NTP, the injected packets are small spoofed requests; the amplification occurs at the reflectors.

- **Amplification attacks with large payloads** (LDAP, MSSQL, Portmap, NetBIOS) use moderate counts (14 M, intensity 2.0), since each injected packet already carries a substantial payload and fewer packets are required to saturate the reflectors.

- **TFTP** receives the lowest count (12 M) and intensity (1.8) because each request triggers a multi-packet server response chain; a lower injection rate is sufficient to produce observable amplified traffic at the detector.

- **WebDDoS** generates two packets per attack unit (a TCP SYN followed by an HTTP GET or POST), so its packet budget is scaled down relative to single-packet flood types at equivalent request rates.

Mixed Attacks

Real-world DDoS campaigns rarely deploy a single attack vector. Sophisticated adversaries combine multiple protocols simultaneously to increase the pressure on the

Table 6.1: Individual attack PCAP generation parameters.

Attack type	Packets	Intensity	Speedup
NTP	20,000,000	3.0	1×
UDP flood	20,000,000	3.0	1×
SYN flood	20,000,000	3.0	1×
DNS	18,000,000	2.5	1×
SNMP	16,000,000	2.5	1×
SSDP	16,000,000	2.0	1×
WebDDoS	15,000,000	2.0	1×
Portmap	14,000,000	2.0	1×
NetBIOS	14,000,000	2.0	1×
LDAP	14,000,000	2.0	1×
MSSQL	14,000,000	2.0	1×
TFTP	12,000,000	1.8	1×

victim and to complicate detection: a classifier trained exclusively on individual attack signatures may encounter ambiguous windows when two or more attack types overlap in feature space, producing a statistical signature that does not clearly belong to any known single class. The four mixed-class PCAPs are designed precisely to expose the model to this scenario during training, forcing it to learn decision boundaries that generalise beyond single-protocol signatures.

Each mixed PCAP contains 25 000 000 packets distributed equally (20% per type) across five complementary attack types, ensuring that no single vector dominates the window statistics and that no two runs share an identical composition. Table 6.2 lists the composition of each run.

Table 6.2: Mixed-class PCAP composition (25M packets each, 20% per type).

Run	Attack types
Mixed 1	DNS, NTP, Portmap, TFTP, SYN
Mixed 2	SNMP, SSDP, NetBIOS, MSSQL, WebDDoS
Mixed 3	DNS, NTP, SSDP, MSSQL, LDAP
Mixed 4	SNMP, Portmap, NetBIOS, TFTP, WebDDoS

Table 6.3 presents the same information as a protocol-presence matrix, making the coverage and complementarity of each run immediately apparent.

The four combinations were selected according to three design criteria:

- **Heterogeneous attack categories.** Mixed 1 pairs high-amplification protocols (NTP ×200, TFTP ×60, DNS ∼41×) with a lower-amplification component (Portmap ×7) and a connection-exhaustion attack (SYN flood). Mixed 4

Table 6.3: Protocol presence matrix for mixed-class PCAPs (✓ = included, 20% share).

	DNS	NTP	SNMP	SSDP	Portmap	NetBIOS	LDAP	MSSQL	TFTP	SYN	WebDDoS
Mixed 1	✓	✓			✓				✓	✓	
Mixed 2			✓	✓		✓		✓			✓
Mixed 3	✓	✓		✓			✓	✓			
Mixed 4			✓		✓	✓			✓		✓
Coverage	2	2	2	2	2	2	1	2	2	1	2

pairs TFTP ×60 with lower-amplification protocols (SNMP ×6, NetBIOS ×4, Portmap ×7) and an application-layer attack (WebDDoS). Mixed 2 similarly combines purely UDP-based amplification with WebDDoS. The presence of TCP flows alongside UDP floods creates a feature-space signature that overlaps with several individual classes, making classification harder.

- **Intra-category overlap.** Mixed 1 is the most challenging run: all five components (DNS, NTP, SSDP, MSSQL, LDAP) are amplification attacks. Their sketch-based features share structural similarities—elevated heavy-hitter counts, high IP concentration, rising per-protocol packet rates—so the window statistics are unlikely to match any individual protocol cleanly. This run specifically tests whether the model can identify a *mixed amplification* scenario without the discriminative contrast provided by a non-amplification component.

- **Full protocol coverage without repetition.** Across the four runs, eleven of the twelve individual attack protocols appear at least once, and no two runs share the same five-protocol set. This diversity ensures that the *mixed* class label encompasses a wide range of multi-vector combinations rather than a narrow cluster in feature space, which would make the class artificially easy to separate from the individual attack classes.

6.1.2 Benign Traffic

Four benign background PCAPs of 25 000 000 packets each were produced using the synthetic traffic generator described in Section 5.1.3. Unlike the attack PCAPs, the benign stream is active throughout the entire 200-second experiment protocol, running concurrently with the attack phase and providing the background traffic that the detector must continuously distinguish from malicious activity. The design of these PCAPs was guided by four requirements:

- **Realistic protocol composition.** Each PCAP draws from four application-level traffic classes: HTTP, DNS, SSH, and UDP. These protocols represent the dominant traffic mix of a typical enterprise or data-centre environment and are sufficient to produce realistic header and payload distributions without introducing unnecessary complexity.

- **Temporal variability within a run.** Each PCAP is structured into four sequential phases of varying duration, each with a distinct protocol distribution and packet-rate multiplier (Table 5.2). The composition shifts from an HTTP-dominant peak (up to 70% of packets) through a DNS-burst phase (up to 50% DNS) and an SSH-stable phase (up to 40% SSH), closing with a UDP-light phase. This intra-run variation reproduces the temporal dynamics of real traffic throughout a workday, preventing the classifier from relying on a static snapshot of benign conditions.

- **Cross-run diversity.** The per-phase distributions and phase durations differ across the four runs: some weight HTTP more heavily, others alternate between DNS and HTTP dominance, and SSH proportions vary substantially. An attack detector trained on a narrow, static benign baseline would be prone to false positives whenever legitimate traffic shifts its composition. Varying the benign profile across runs forces the model to learn that temporal fluctuations in protocol mix are normal, not indicative of an attack.

- **Escalating line rates.** Each PCAP is replayed at a progressively higher target rate across runs (8, 9, 10, and 12 Gbps), ensuring that the detector is exercised under different absolute traffic loads. The full rate configuration is reported in Section 6.2.1.

6.1.3 Experiment Protocol

Each data collection run follows a fixed 200-second three-phase protocol designed to provide clean ground-truth labels for supervised learning. Figure 6.1 illustrates the timing structure.

- **Benign baseline ($t = 0$–$50\,\mathrm{s}$).** Only the controller node transmits traffic. This phase establishes a clean record of normal traffic behaviour under the specific benign PCAP and line rate selected for the run. The detector observes no attack traffic during this interval, and all detection windows falling within it are labelled `benign`.

- **Attack phase ($t = 50$–$150\,\mathrm{s}$).** The attacker node begins replaying its PCAP concurrently with the ongoing benign stream. The combined traffic is received

Figure 6.1: Data collection run protocol: 50 s benign baseline, 100 s attack phase, 50 s benign recovery.

by the monitor throughout this 100-second window, exposing the detector to the target attack type at the configured rate while background traffic remains active. All detection windows in this interval are labelled with the attack class of the active PCAP.

- **Benign recovery** ($t = 150$–200 s). The attack sender stops and only benign traffic continues. This phase captures the transient period immediately after an attack ceases and confirms that the detector returns to a normal operating state. Windows in this interval are labelled `benign`.

The labelling procedure is applied offline during dataset construction and is detailed in Section 6.2.

6.2 Dataset Construction

The dataset is built from a total of 13 attack classes × 4 intensity runs = 52 experimental runs, covering 12 individual attack types plus a mixed class. In every run, benign background traffic runs concurrently throughout the full 200-second protocol (as the baseline and recovery phases described in Section 6.1.3); it is not a separate set of runs. The detector logs from each run are parsed offline to produce two parallel feature datasets (one in DPI-Sketch mode and one in Sketch-ADV mode) extracted from the same logs and therefore directly comparable.

6.2.1 Replay Rate Configuration

Benign Replay

Table 6.4 summarises the benign configuration for each paired run, including the dominant protocol phase, line rate, and inter-packet jitter.

Table 6.4: Benign PCAP configuration per run.

Run	Gbps	Dominant phases	Jitter
Benign 1	8	HTTP peak / DNS burst	0 ms
Benign 2	9	HTTP/DNS balanced	5 ms
Benign 3	10	HTTP heavy / SSH spike	10 ms
Benign 4	12	Mixed HTTP/DNS / SSH	15 ms

The jitter increases progressively from 0 ms to 15 ms across runs, spanning the full range supported by the benign sender. This variation ensures that the combined traffic presented to the detector during each run has a distinct temporal texture: run 1 produces a smooth, zero-jitter benign stream, while run 4 introduces moderate inter-packet variability that challenges the sketch-based temporal features.

Attack Replay

Each attack PCAP is replayed at four different target line rates (run1–run4), and each run is paired with the corresponding benign PCAP at its associated line rate and jitter setting. The progressive increase across runs is deliberate: it exposes the model to a range of attack intensities, from low-rate injection to near-saturation conditions, improving generalisation to real-world scenarios where attack volume is not fixed. Table 6.5 lists the replay rates for each attack type and run.

Table 6.5: Attack PCAP replay rates (Gbps) per run.

Attack type	Run 1	Run 2	Run 3	Run 4
NTP	2.5	2.0	3.0	3.5
DNS	1.8	2.0	2.2	2.5
SNMP	1.4	1.8	2.2	2.6
SSDP	2.2	2.4	2.6	2.8
Portmap	1.4	1.8	2.2	2.6
NetBIOS	1.8	2.0	2.2	2.5
LDAP	1.4	1.8	2.2	2.6
MSSQL	0.8	1.0	1.2	1.4
UDP flood	10.0	11.0	12.0	13.0
SYN flood	1.2	1.4	1.6	1.8
WebDDoS	2.8	3.0	3.2	3.4
TFTP	0.6	0.8	1.2	1.4
Mixed	2.5	3.0	3.5	4.0

The rate assignments reflect the traffic profile of each attack category:

- **Reflection/amplification attacks** (NTP, DNS, SNMP, SSDP, Portmap, NetBIOS, LDAP, TFTP): replayed at 0.6–3.5 Gbps. Attack effectiveness derives

from the amplification factor at the reflector, not from the raw bandwidth of the injected requests: a modest stream of small requests triggers a disproportionately large response volume at the victim. MSSQL sits at the lowest end (0.8–1.4 Gbps) since its ping request payload is a single byte.

- **UDP flood**: replayed at 10–13 Gbps, the highest rates in the table. This is a direct bandwidth exhaustion attack whose sole objective is to saturate the link. The upper bound is constrained by the 25 Gbps link capacity: at run 4, UDP flood (13 Gbps) plus benign traffic (12 Gbps) reaches 25 Gbps, exactly at the link limit.

- **SYN flood**: replayed at 1.2–1.8 Gbps. This attack targets connection-table exhaustion rather than bandwidth each packet is a bare TCP SYN with no payload (≈40 bytes), so even a high packet rate translates to modest Gbps.

- **WebDDoS**: replayed at 2.8–3.4 Gbps. Each attack unit emits two packets (a TCP SYN followed by an HTTP GET/POST request with payload), yielding higher bandwidth than a pure SYN flood at similar packet rates.

- **Mixed**: replayed at 2.5–4.0 Gbps, averaging the constituent attack types of each mixed PCAP and therefore falling in the intermediate range.

Figure 6.2 shows the combined link load observed at the monitor node across all 52 runs, split by attack category. Each group of four bars represents one attack class, with bars increasing from run 1 (lowest intensity) to run 4 (highest intensity). The following points summarise the key traffic dynamics:

- **Benign baseline** (blue) occupies 8–12 Gbps in every run, growing progressively as runs are paired with heavier benign PCAPs. This persistent background ensures the classifier is always trained on mixed traffic rather than isolated attack streams.

- **Reflection/amplification attacks** (amber, top panel) are injected at modest rates of 0.6–3.5 Gbps, far below the link limit at the monitor. However, their amplification factors range from ×4 (NetBIOS) to ×200 (NTP), meaning the reflectors return a response volume orders of magnitude larger than the original request. The estimated victim-side traffic therefore exceeds the 25 Gbps uplink for most runs, as indicated by the red hatching: five protocols (NTP, DNS, SSDP, LDAP, TFTP) saturate the victim's link in all four runs, while MSSQL (×25) reaches the link limit from run 2. SNMP (×6), Portmap (×7), and NetBIOS (×4) deliver at most 16, 18, and 10 Gbps respectively to the victim and do not saturate the link under the configured injection rates.

- **UDP flood** (orange, bottom panel) is the only attack that challenges the *monitor* link rather than the victim's: at run 4, the 13 Gbps injection combined with 12 Gbps of benign traffic reaches exactly the 25 Gbps physical limit. No amplification is involved; the attack effectiveness is purely a function of injected volume.

- **SYN flood** and **WebDDoS** remain well below 25 Gbps at the monitor (1.8 Gbps and 3.4 Gbps at run 4 respectively). Their impact on the victim is resource-based, not bandwidth-based: SYN flood exhausts the kernel's half-open connection table, while WebDDoS saturates the web server's request queue. Neither creates a link-level bottleneck on a 25 Gbps testbed, so no hatching is shown.

- **Mixed attacks** contain amplification components in every PCAP (e.g. DNS, NTP, SNMP, TFTP); even at modest injection rates the amplified sub-components collectively saturate the victim's link across all runs, hence the red hatching throughout.

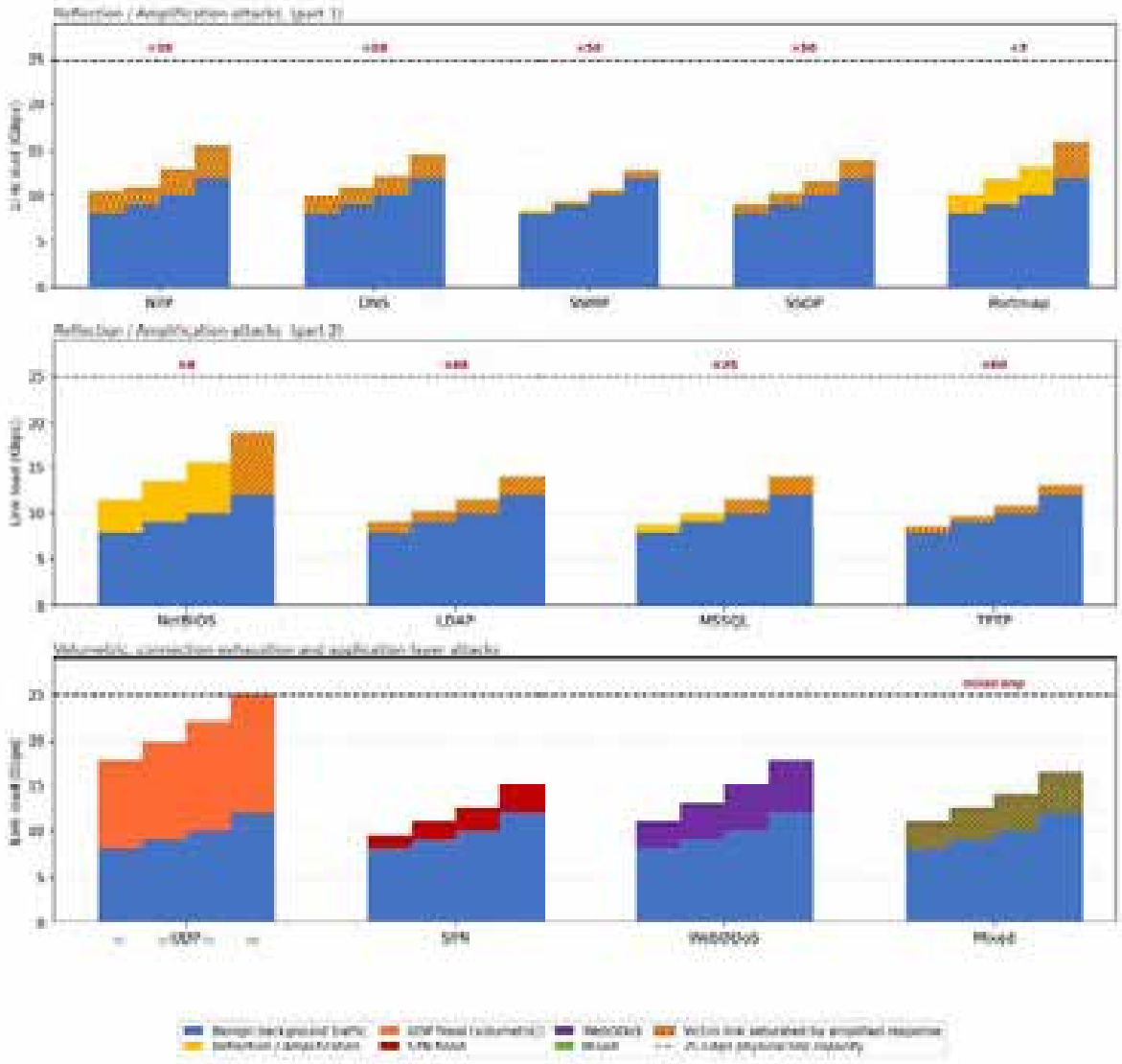

Figure 6.2: Monitor-link load across all 52 experimental runs.

6.2.2 Feature Extraction and Labelling

Each experimental run is executed twice: once with the detector operating in DPI-Sketch mode and once in Sketch-ADV mode. Both runs use identical traffic (same PCAP pair and line rates), so the two resulting feature datasets are directly comparable. The detector logs from each run are parsed by an offline feature extractor that reads the per-window statistics and produces one CSV row per 50 ms window.

Labels are assigned based on the elapsed time within the run: windows in [0s, 50s) and [150s, 200s) receive label `benign`; windows in [50s, 150s) receive the attack class of the active PCAP (e.g. `ntp`, `syn`, `mixed_1`). The label assignment requires no manual annotation and is fully reproducible from the run metadata.

The final dataset for each mode is assembled by concatenating all labelled CSVs, ap-

plying standard scaling (zero mean, unit variance) fitted on the training split, and partitioning into training (70%), validation (15%), and test (15%) sets with stratification by class label to preserve the class distribution across splits.

Table 6.6 reports the per-class sample counts for each split. Almost the same partition is applied to both DPI-Sketch and Sketch-ADV: both use a 70/15/15 stratified split, but since each mode may produce a slightly different number of extracted windows from the same experimental runs, the absolute sample counts differ marginally between modes.

Table 6.6: Per-class sample distribution across training, validation, and test splits.

Class	Total	Train (70%)	Val (15%)	Test (15%)
Benign	61,033	42,723	9,155	9,155
DNS	3,007	2,105	451	451
LDAP	3,220	2,254	483	483
Mixed	2,620	1,834	393	393
MSSQL	3,253	2,277	488	488
NetBIOS	3,220	2,254	483	483
NTP	3,013	2,109	452	452
Portmap	3,220	2,254	483	483
SNMP	3,153	2,207	473	473
SSDP	3,147	2,203	472	472
SYN	3,020	2,114	453	453
TFTP	3,307	2,315	496	496
UDP	2,940	2,058	441	441
WebDDoS	2,973	2,081	446	446
Total	**101,120**	**70,788**	**15,169**	**15,169**

The class distribution is strongly imbalanced: benign samples account for 60.4% of the dataset. The imbalance has two compounding causes. First, benign windows accumulate across all 52 experimental runs, because every run includes both a baseline and a recovery phase regardless of the attack class being tested. Each attack class, by contrast, appears in only 4 of the 52 runs (one per intensity level), so its attack windows are drawn from 4 runs while the benign pool draws from all 52. This alone produces a theoretical benign-to-attack ratio of $52/4 = 13$, since each 200-second run contributes approximately the same number of benign windows (100 s at 50 ms per window) as attack windows. Second, the 100-packet-per-window threshold filters out a larger fraction of attack windows than benign windows: high-rate benign traffic (8–12 Gbps) generates thousands of packets per window and is almost never filtered, whereas low-rate attacks such as MSSQL (0.8 Gbps) and TFTP (0.6 Gbps) produce fewer than 100 packets in the initial windows of the attack phase, causing those windows to be discarded. The combination of these two effects raises the observed ratio to approximately 20:1 (61 033 benign vs. ≈3 000 per attack class). Among attack classes,

sample counts range from 2,620 (Mixed) to 3,307 (TFTP), reflecting minor variation in the number of windows that survive the packet threshold across different attack rates. The stratified split preserves this imbalance across all three partitions, so the test set evaluates models under the same class distribution as training.

6.3 Model Training and Comparison

Seven supervised classifiers spanning four algorithmic families were evaluated on both feature modes. The evaluation protocol is identical for every candidate: each model is trained on the training split, selected on the validation split, and scored on the held-out test split. All feature vectors are standardised with a `StandardScaler` fitted exclusively on the training data before being passed to any model.

6.3.1 Candidate Models

The candidate set was designed to cover a representative range of inductive biases and computational profiles. Each model was selected either as a strong baseline, as a representative of its algorithmic family, or because of a specific property relevant to deployment at line rate.

- **Random Forest.** Included as the canonical ensemble baseline for tabular classification. A Random Forest builds an ensemble of T independent decision trees, each trained on a bootstrap sample of the training data with a random subset of features considered at each split. Predictions are obtained by majority vote across trees. The algorithm is naturally robust to irrelevant features and to moderate class imbalance, and it requires no feature scaling. The configuration used here is $T = 200$ trees with maximum depth 8 and minimum leaf size 5, a setting that limits overfitting while retaining sufficient capacity for 14-class separation.

- **Histogram Gradient Boosting.** Included as a representative of modern boosted tree methods. Unlike the Random Forest, which builds trees independently, gradient boosting constructs trees sequentially: each tree fits the negative gradient of the loss of the current ensemble, progressively correcting its residual errors. The histogram variant bins continuous features into discrete buckets before finding splits, reducing both training time and memory usage compared to exact-split gradient boosting. With 200 iterations, maximum depth 6, and minimum leaf size 10, it typically dominates plain bagging methods on structured tabular data with heterogeneous feature scales, which is precisely the character of the sketch-based feature vectors used here.

- **Multilayer Perceptron.** Included to assess whether a feed-forward neural net-

work can exploit non-linear interactions among features that tree methods may miss. The architecture is a two-hidden-layer network with 128 and 64 rectified linear units, trained end-to-end with Adam and L2 weight decay. Early stopping on the validation loss prevents overfitting on the smaller attack classes such as *mixed* and *webddos*.

- **K-Nearest Neighbours.** Included as a non-parametric reference that makes no assumption about the functional form of the decision boundary. Classification is performed by a majority vote among the $k = 5$ nearest training samples in the scaled feature space. KNN is sensitive to the curse of dimensionality, so its performance on the 75-feature DPI-Sketch mode is expected to degrade relative to the 64-feature Sketch-ADV mode. Its primary role in this evaluation is to establish a geometry-based lower bound against which parametric models can be compared.

- **SGD Classifier.** Included as the fast linear baseline. The model fits a linear discriminant in the scaled feature space using stochastic gradient descent with the modified Huber loss, which combines the robustness of hinge loss with probabilistic output calibration. If the attack classes are approximately linearly separable after standardisation, this model will perform competitively at a fraction of the training cost of ensemble or deep methods. A poor result on the linear baseline motivates the use of more expressive models.

- **LSTM.** Included to represent recurrent architectures capable of modelling temporal dependencies across consecutive detection windows. The configuration is a two-layer LSTM with hidden size 128 and dropout 0.3, trained for up to 100 epochs with a patience of 10 on the validation loss. Although each window is presented as a single time step in this experiment, the recurrent cell still learns a richer internal representation than a feed-forward layer of the same width, because the gating mechanism allows it to selectively suppress or amplify individual input dimensions rather than treating all features symmetrically.

- **LightGBM.** Included as the primary deployment candidate. LightGBM is a gradient-boosted decision tree framework that applies two key optimisations over standard gradient boosting: Gradient-based One-Side Sampling reduces the number of data points considered at each iteration by retaining only those with large gradients, and Exclusive Feature Bundling groups mutually exclusive sparse features to reduce the effective dimensionality. The result is faster training and lower memory consumption than comparable boosted tree libraries. Beyond its accuracy, LightGBM is the only candidate in the evaluation that exposes a production C API, enabling the serialised model to be loaded and invoked from C

code at inference time without any Python dependency.

6.3.2 Training Procedure

Feature Standardisation

All models except Random Forest and LightGBM receive inputs that have been standardised to zero mean and unit variance using a `StandardScaler` fitted exclusively on the training split. The scaler is never re-fitted on the validation or test data; instead, the training-set statistics (per-feature mean μ_i and standard deviation σ_i) are applied as fixed linear transformations to all subsequent partitions:

$$\tilde{x}_i = \frac{x_i - \mu_i}{\sigma_i}, \tag{6.1}$$

This ensures that no information from the validation or test distributions leaks into the normalisation step. Random Forest and LightGBM are scale-invariant by construction (their split decisions depend only on feature ordering, not on absolute magnitude), but they receive the standardised features as well, for a fair comparison with distance-sensitive models such as KNN.

The two feature modes are treated as entirely independent training campaigns: DPI-Sketch and Sketch-ADV each maintain their own scaler fitted on their own training split, so the per-feature statistics of one mode do not influence the other.

Multi-class Strategy

The classification problem has 14 mutually exclusive classes (benign plus 13 attack types). Each model handles this natively:

- **Random Forest, HistGradientBoosting, KNN, SGD.** The scikit-learn implementations extend intrinsically to multi-class via a one-vs-rest strategy for SGD and native multi-output splits for the tree-based methods. No class decomposition is required.
- **MLP and LSTM.** The output layer consists of 14 softmax-normalised units, one per class. The training objective is categorical cross-entropy over the full 14-class label distribution.
- **LightGBM.** The `objective` is set to `multiclass` (softmax), with `num_class = 14`. The evaluation metric used for early stopping is `multi_logloss`, which penalises confident wrong predictions more severely than accuracy, encouraging well-calibrated probability outputs across all classes.

Class Imbalance

The dataset exhibits a pronounced class imbalance: benign samples account for 60.4% of the training data, while the smallest attack class (Mixed, 1,834 training samples) represents only 1.8%. A naïve classifier that always predicts benign would achieve 60% accuracy, making raw accuracy an unreliable primary metric.

Two mechanisms mitigate this imbalance. First, `class_weight=''balanced''` is enabled for all scikit-learn classifiers, automatically rescaling the per-sample loss by the inverse class frequency so that each class contributes equally to the training gradient regardless of its size. Second, the primary ranking metric used throughout the evaluation is the weighted F1 score, which weights each class's F1 by its support in the test set and is therefore sensitive to both precision and recall across the full class distribution, not just on the dominant benign class. Macro F1 is also reported where relevant to assess average per-class performance without frequency weighting. LightGBM does not use the `class_weight` parameter directly; instead, the `is_unbalance` flag is set, which internally rescales the gradient updates to compensate for unequal class frequencies.

Model Selection and Early Stopping

The validation split serves two distinct purposes: hyperparameter selection and, for iterative models, early stopping.

For scikit-learn models, training is performed in a single pass with fixed hyperparameters. The validation set is used solely to compare candidate configurations and select the best-performing variant.

For the MLP and LSTM, early stopping monitors the validation loss (cross-entropy) after each epoch and halts training if no improvement is observed for 10 consecutive epochs. The model weights at the epoch with the lowest validation loss are restored before test evaluation.

For LightGBM, the native `early_stopping_rounds` parameter instructs the booster to stop adding trees if the validation `multi_logloss` does not decrease for 50 consecutive rounds. This resulted in 500 trees on the DPI-Sketch complete dataset and 499 trees on the Sketch-ADV complete dataset. The final model checkpoint at the best iteration is retained for all subsequent evaluations.

Table 6.7 summarises the hyperparameter configuration for each candidate. All settings were selected by manual search on the validation split; no automated optimisation was applied.

Table 6.7: Hyperparameter configuration for each candidate model.

Model	Key hyperparameters
Random Forest	200 trees; max depth 8; min samples per leaf 5
HistGradientBoosting	200 iterations; max depth 6; min samples per leaf 10
MLP	Hidden layers [128, 64] (ReLU); Adam, $\eta = 10^{-3}$; L2 weight decay 10^{-4}; early stopping on val loss
KNN	$k = 5$; Euclidean distance; uniform weights
SGD	Modified Huber loss; $\alpha = 10^{-4}$; max 1000 iterations; tolerance 10^{-3}
LSTM	2 layers; hidden size 128; dropout 0.3; Adam $\eta = 10^{-3}$; patience 10 epochs
LightGBM	500 trees (DPI-Sketch); 499 trees (Sketch-ADV); learning rate 0.1; early stopping on val set

6.3.3 Model Comparison Results

Table 6.8 reports test-set accuracy and weighted F1 score for all seven candidates on both feature modes. DPI-Sketch results are from the complete dataset (75 features, 70,788 training samples); Sketch-ADV results are from the complete dataset (64 features, 72,959 training samples). Best values in each column are shown in bold; † marks the model selected for production deployment.

Table 6.8: Model comparison across both feature modes. Best per-column in bold; † = selected for deployment.

	DPI-Sketch (75 feat.)		Sketch-ADV (64 feat.)	
Model	**Acc**	**wF1**	**Acc**	**wF1**
Linear / distance-based baselines				
SGD	96.53%	96.24%	98.04%	99.03%
KNN	99.08%	99.03%	97.51%	97.31%
Tree-based ensembles				
Random Forest	98.44%	98.22%	**98.85%**	98.82%
LightGBM†	99.82%	99.81%	98.66%	98.54%
HistGradientBoosting	**99.91%**	**99.91%**	98.43%	99.42%
Neural networks				
LSTM	99.76%	99.76%	98.62%	**99.61%**
MLP	99.89%	99.89%	97.24%	99.24%

Overall picture. With the exception of SGD on DPI-Sketch, every model surpasses 97% weighted F1 on both modes, confirming that the sketch-based feature vectors

provide a strong signal across all algorithmic families. The top tier (HistGradientBoosting, MLP, LightGBM, and LSTM) clusters tightly between 99.76% and 99.91% wF1 on DPI-Sketch, a gap of only 0.15 percentage points. On Sketch-ADV the spread is wider (97.31%–99.61%), reflecting the harder classification problem that arises without DPI payload counters and with a smaller training set.

Linear and distance-based baselines. SGD is the weakest model on DPI-Sketch (96.24% wF1) but ranks among the better performers on Sketch-ADV (99.03% wF1). This inversion is explained by the feature structure of each mode: the 75 DPI-Sketch features include raw protocol counters, heavy-hitter estimates, and derived ratios whose decision boundaries are fundamentally non-linear, preventing a linear discriminant from achieving a tight fit. The 64 Sketch-ADV features are dominated by `ratio_vs_total_*` values that express each protocol's fractional share of traffic; these ratios are more linearly separable because a clean single-protocol attack drives one ratio close to 1 while all others collapse to near 0.

KNN exhibits the opposite trend: strong in DPI-Sketch (99.03% wF1, rank 5) but the weakest tree/neural competitor on Sketch-ADV (97.31%, last place). The explanation lies in the geometry of the Sketch-ADV feature space. Its 64 features are dominated by `ratio_vs_total_*` values that express each protocol's fractional share of traffic. In this ratio-dominated space, multiple attack classes can produce near-identical Euclidean distances to a query point when several protocols are simultaneously elevated, as occurs during mixed or blended attack windows. Tree-based and neural models resolve these boundary ambiguities through learned non-linear splits or weighted combinations, but KNN's majority vote among the five nearest neighbours breaks down when the local neighbourhood is populated by samples from geometrically close but semantically distinct classes.

Tree-based ensembles. Random Forest, LightGBM, and HistGradientBoosting form a consistent group across both modes, all remaining above 98.5% wF1 on Sketch-ADV. On DPI-Sketch, HistGradientBoosting (#1, 99.91% wF1) and LightGBM (#3, 99.81%) lead the group, while Random Forest ranks lower (#6, 98.22% wF1) — a gap that reflects its weaker handling of non-linear DPI feature interactions compared to gradient-boosted trees. Random Forest is the only tree-based model to achieve higher accuracy on Sketch-ADV than DPI-Sketch, reflecting its robustness to small training sets thanks to bootstrap aggregation.

Neural networks. MLP and LSTM both reach the top tier on DPI-Sketch (99.89% and 99.76% wF1 respectively) but show different behaviour on Sketch-ADV. The LSTM achieves the highest wF1 of all models on Sketch-ADV (99.61%), suggesting that its

gated memory mechanism exploits temporal structure in the sketch-only feature space better than the feedforward MLP. The MLP, despite its high wF1 (99.24%), shows the largest accuracy–wF1 gap on Sketch-ADV (97.24% acc vs 99.24% wF1), indicating that it misclassifies a disproportionate fraction of the majority benign class while maintaining near-perfect performance on the minority attack classes.

The *mixed* class as a stress test. Across all models and both modes, the *mixed* class consistently records the lowest per-class F1 score. Its windows blend the sketch statistics of five concurrent attack protocols, creating a feature-space signature that overlaps with each of its constituent individual-attack classes. On DPI-Sketch, LightGBM achieves F1 = 0.974 on mixed, close to the per-class mean; on Sketch-ADV it drops to 0.750, the most pronounced single-class degradation observed in the entire evaluation. This underlines that the DPI payload counters are not merely additive features but provide qualitatively different discriminative information that cannot be recovered from sketch aggregates alone when multiple attack signals co-exist.

6.3.4 LightGBM Selection

The selection of a production classifier cannot be reduced to a single accuracy metric. The DPDK fast path imposes a set of non-negotiable engineering constraints that are entirely orthogonal to statistical performance: the model must be callable from C without any external runtime, its inference latency must be deterministic and bounded, it must be stateless so that multiple worker threads can invoke it concurrently without synchronisation, and it must integrate into a codebase that already manages hardware queues, memory pools, and NUMA-aware packet buffers. Table 6.9 evaluates all seven candidates against these deployment criteria in addition to accuracy.

Table 6.9: Deployment suitability of each candidate model. ✓= fully satisfied, ∼ = partially, × = not satisfied.

Model	C API	No runtime	Bounded latency	Thread-safe	Top-4 acc.
SGD	×	×	✓	✓	×
KNN	×	×	×	✓	×
Random Forest	×	×	✓	∼	×
LightGBM	✓	✓	✓	✓	✓
HistGradientBoosting	×	×	✓	∼	✓
MLP	×	×	∼	✓	✓
LSTM	×	×	∼	×	✓

LightGBM is the only model that satisfies all five criteria simultaneously. The remaining subsections detail each criterion and explain why the higher-accuracy alternatives cannot be used in this deployment context.

Native C Inference API

LightGBM is the only candidate in the evaluation set that ships a production-grade C inference API as part of its official distribution. The trained model is serialised to a human-readable text file and loaded at runtime using a single call:

```
LGBM_BoosterLoadModelFromString(model_str, &booster);
```

Inference on a single feature vector is then performed via `LGBM_BoosterPredictForMat`, returning a 14-element probability array with no dependency on Python, a JVM, or any shared library beyond `liblightgbm.so`. This API is usable from any C or C++ compilation unit and is binary-compatible with the DPDK poll-mode driver threads.

Every other candidate requires a language runtime that is fundamentally incompatible with a DPDK fast path:

- **HistGradientBoosting, Random Forest, KNN, SGD** are implemented in scikit-learn and serialised via Python's `pickle` protocol. Invoking any of them requires a live CPython interpreter, a `libpython` shared object, and the full NumPy/SciPy stack, none of which can be safely loaded into a DPDK lcore thread that runs in poll mode and may never block.
- **MLP and LSTM** are implemented in PyTorch, which requires the `libtorch` runtime (several hundred megabytes), CUDA initialisation overhead if a GPU is present, and an `at::Tensor` heap allocator that is not compatible with DPDK's `rte_mempool` object model.

Deterministic Bounded Latency

In a DPDK fast path, the classifier is invoked every 50 ms by a dedicated lcore thread. The lcore must return control to the packet-processing loop promptly and must never stall for an unpredictable duration.

LightGBM inference consists of traversing $T = 500$ independent shallow decision trees, each of depth ≤ 6. Every tree traversal is a sequence of at most 6 branch comparisons against pre-computed thresholds stored in a contiguous array. The total operation count is fixed and data-independent: regardless of the input feature vector, exactly the same number of array accesses and comparisons is performed. This makes LightGBM inference $O(1)$ in the worst case with a very small constant, and its latency profile is as deterministic as a lookup table.

Neural network inference, by contrast, involves floating-point matrix multiplications whose execution time varies with CPU frequency scaling and cache pressure. The

LSTM additionally maintains a hidden-state vector that must be zeroed and re-initialised at each inference call if windows are treated as independent (as they are here), adding an $O(h)$ memset operation and breaking the constant-time guarantee. MLP inference is more predictable but still dominated by BLAS matrix-vector products that interact with the NUMA memory hierarchy in ways that are hard to bound under interrupt pressure.

KNN inference requires scanning all $N = 70{,}788$ training vectors to find the k nearest neighbours, making it $O(N \cdot d)$ per classification call, completely incompatible with a real-time constraint.

Thread Safety and Stateless Inference

The detector runs 14 worker threads in parallel on a 16-core server. Each worker independently processes packets from its assigned NIC receive queue, accumulates sketch statistics into its own per-worker data structure, and at the end of each 50 ms window invokes the classifier on the resulting feature vector. For correctness, classifier inference must be reentrant: two threads must be able to call the classifier simultaneously on different feature vectors without interfering with each other.

LightGBM inference is fully stateless. The booster object loaded via the C API is read-only after model loading. it contains only the pre-trained tree parameters and threshold arrays, which are never modified during inference. Any number of threads can call `LGBM_BoosterPredict` concurrently on the same booster handle with different input arrays and separate output buffers, with no locking required.

The LSTM, by contrast, maintains a hidden-state vector (h_t, c_t) that is updated at each inference step. Even if individual inference calls treat each window independently and reset the state to zero, the hidden-state buffer itself must be per-thread, requiring either one LSTM instance per lcore (multiplying memory usage by 14) or a thread-local storage mechanism that adds synchronisation overhead incompatible with the DPDK programming model.

Accuracy in Context

LightGBM's measured accuracy (99.82% on DPI-Sketch, 98.66% on Sketch-ADV) is not the highest in the evaluation. HistGradientBoosting (99.91%) and MLP (99.89%) both exceed it on DPI-Sketch by 0.09 and 0.07 percentage points respectively (roughly 14 and 11 additional correctly classified samples in the 15,169-sample test set).

This gap is accepted for four reasons:

1. **Margin is negligible at operational scale.** At 37 Mpps line rate and 50 ms windows, each classification decision covers roughly 1.85 million packets. A 0.09-point accuracy difference represents fewer than two misclassified windows per 2,000, a false alarm rate that is well within the acceptable bounds of a production DDoS detector.

2. **No other model satisfies the deployment criteria.** The models that outperform LightGBM (HistGBM, MLP) fail the C API requirement unconditionally. No engineering workaround (ONNX export, model distillation, or custom C reimplementation) can replicate their exact trained parameters without substantial additional effort and without the risk of numerical discrepancies introduced by the translation.

3. **LightGBM is the only deployable model with cross-mode consistency.** While HistGBM and MLP both rank in the top three on DPI-Sketch and exceed 98.5% wF1 on Sketch-ADV, neither satisfies the C API requirement and therefore cannot be used without a Python runtime. LightGBM is the only model that meets all deployment constraints *and* maintains top-three accuracy on DPI-Sketch with above 98.5% wF1 on Sketch-ADV. This cross-mode consistency is valuable in a system that may switch between feature modes at runtime depending on hardware capabilities.

4. **LightGBM training is fast and operationally practical.** A full retraining on either feature mode completes in under three minutes on a single CPU core, with no GPU dependency. This makes periodic retraining feasible as traffic patterns evolve, without requiring a dedicated training infrastructure. By contrast, the MLP and LSTM require GPU acceleration to achieve comparable training times, and their training pipelines depend on a Python runtime that is incompatible with the production deployment environment.

6.3.5 LightGBM Mode Comparison

To isolate the contribution of each feature mode, LightGBM was trained and evaluated independently on both DPI-Sketch and Sketch-ADV using their respective complete splits. Table 6.10 summarises the overall metrics and Table 6.11 reports the per-class F1 score for each mode.

DPI-Sketch consistently outperforms Sketch-ADV on nearly every class, with the most pronounced gap on the *mixed* class: F1 drops from 0.974 to 0.750 when switching to Sketch-ADV. This is explained by the DPI features (which include per-protocol request counters such as `ldap_search_requests`, `portmap_getport_calls`, and `mssql_-`

Table 6.10: LightGBM overall metrics by feature mode.

Mode	Features	Accuracy	Weighted F1
DPI-Sketch	75	99.82%	99.81%
Sketch-ADV	64	98.66%	98.54%

Table 6.11: LightGBM per-class F1 score by feature mode.

Class	DPI-Sketch	Sketch-ADV
Benign	1.000	0.998
DNS	0.993	1.000
LDAP	0.997	0.944
Mixed	0.974	0.750
MSSQL	0.999	0.971
NetBIOS	0.996	1.000
NTP	0.997	0.971
Portmap	0.995	1.000
SNMP	1.000	1.000
SSDP	0.995	1.000
SYN	1.000	0.970
TFTP	1.000	1.000
UDP	1.000	0.968
WebDDoS	0.997	0.968
Macro avg	0.996	0.967

`sqlbatch_packets`) providing fine-grained protocol-level signals that allow the classifier to distinguish overlapping attack mixtures. The Sketch-ADV feature set lacks these per-packet payload signatures and relies instead on aggregate flow statistics, making it harder to separate windows where multiple attack protocols co-exist.

Classes that depend on volumetric or port-based signals, such as SNMP, SSDP, Portmap, TFTP, and DNS, reach perfect or near-perfect F1 on both modes, confirming that sketch-based counters are sufficient to detect attacks with a dominant bandwidth or packet-rate signature. The gap between modes therefore narrows for simple volumetric floods and widens for protocol-specific or mixed-composition attacks.

The top discriminative features identified by LightGBM differ markedly between modes. For DPI-Sketch, the five most important features are `top_ip_pps_50ms`, `ip_concentration`, `ldap_search_requests`, `portmap_getport_calls`, and `mssql_sqlbatch_packets` (a combination of heavy-hitter counters and DPI-derived protocol request rates). For Sketch-ADV, the top five are `pps_baseline`, `adaptive_threshold`, `top_ip_pps_1s`, `ratio_vs_total_portmap`, and `ratio_vs_total_snmp` (purely statistical features derived from sketch aggregates). This confirms that the two modes exploit fundamentally different signal types and are complementary rather than redundant.

6.3.6 Feature Importance Analysis

LightGBM records a split-based importance score for each input feature, counting the total number of times that feature is chosen as a split criterion across all trees in the ensemble. Tables 6.12 and 6.13 list the top-15 features for each mode.

Table 6.12: Top-15 LightGBM feature importances, DPI-Sketch mode (complete dataset, 75 features).

Rank	Feature	Importance
1	top_ip_pps_50ms	589,833
2	ip_concentration	285,703
3	ldap_search_requests	231,246
4	portmap_getport_calls	168,130
5	mssql_sqlbatch_packets	156,225
6	ntp_monlist_queries	142,681
7	netbios_name_queries	135,812
8	ssdp_msearch_packets	135,356
9	snmp_ratio	134,159
10	tftp_ratio	128,307
11	attack_packets	122,638
12	active_attack_protocols	114,746
13	dns_any_queries	107,770
14	mssql_ratio	99,918
15	syn_only_ratio	95,741

Both tables are trained on datasets of equivalent size (70,788 and 72,959 training samples respectively), so their split-based importance scores are directly comparable in magnitude. The top feature in each mode reaches nearly identical absolute importance: `top_ip_pps_50ms` at 589,833 for DPI-Sketch and `pps_baseline` at 575,077 for Sketch-ADV, and the overall concentration is also similar: the rank-1 to rank-15 ratio is ∼6.2× for DPI-Sketch and ∼5.0× for Sketch-ADV. The principal difference between the two profiles therefore lies not in their overall shape but in the nature of the features that dominate.

In DPI-Sketch, the classifier's primary split is on `top_ip_pps_50ms` (heavy-hitter rate at 50 ms), followed by `ip_concentration`. Ranks 3–8 are exclusively DPI-derived protocol request counters: `ldap_search_requests`, `portmap_getport_calls`, `mssql_sqlbatch_packets`, `ntp_monlist_queries`, `netbios_name_queries`, and `ssdp_msearch_packets`, each directly counting the volume of protocol-specific traffic in the window. This ordering reveals a two-stage decision structure: first identify that traffic is anomalously concentrated from a small set of sources, then use the DPI payload counters to determine *which* reflection protocol is responsible. Notably, 15 of the 75 features recorded zero importance, including `fragmentation_ratio`, `snmp_amplification_factor`, and

Table 6.13: Top-15 LightGBM feature importances, Sketch-ADV mode (complete dataset, 64 features).

Rank	Feature	Importance
1	pps_baseline	575,077
2	pps_variance	262,463
3	ratio_vs_total_mssql	218,557
4	ratio_vs_total_http	189,075
5	ratio_vs_total_snmp	182,542
6	ratio_vs_total_netbios	179,144
7	ratio_vs_total_ntp	164,709
8	ratio_vs_total_ssdp	154,766
9	ratio_vs_total_portmap	145,773
10	ratio_vs_total_tftp	137,983
11	pps_udp_other	137,306
12	ratio_vs_total_ldap	132,803
13	adaptive_threshold	131,960
14	ratio_vs_total_dns	116,404
15	ratio_vs_total_syn	114,497

ntp_amplification_factor, confirming that these derived ratios are entirely redundant given the raw protocol counters already present.

In Sketch-ADV, the top two features are pps_baseline (absolute packet rate at steady state) and pps_variance (temporal variability of the packet rate), which together capture whether traffic has deviated from its expected level and how erratically it is fluctuating. Ranks 3–15 are dominated by eleven ratio_vs_total_* features, each expressing one protocol's share of the total observed packet rate, with no individual protocol dramatically outweighing the others. Without DPI payload counters to pinpoint a specific reflection protocol, the model distributes its discriminative signal across the full set of per-protocol ratio features. This is consistent with the degraded performance on the *mixed* class: when multiple protocols are simultaneously elevated, no single ratio_vs_total_* feature provides a clean split, and the classifier must rely on joint conditions across many features to reach a confident prediction.

Chapter 7

Evaluation and Discussion

The preceding chapters described the design of a line-rate DDoS detection system and the experimental methodology used to evaluate it. This chapter steps back from the individual numbers to ask whether the system meets the objective stated in Section 4.2: a detector capable of operating in high-speed and partially encrypted network environments while maintaining competitive classification performance. The central question is whether replacing DPI payload inspection with purely header-derived sketch features imposes an accuracy penalty that is operationally unacceptable, or whether the resulting system strikes a useful balance between classification fidelity and the structural requirements of a real deployment.

7.1 Comparative Performance: Sketch-ADV Approaches DPI-Sketch

The sketch-based architecture described in Chapter 4 was motivated by bounded memory efficiency: replacing the per-flow state tables of traditional DPI systems with fixed probabilistic structures capable of sustaining high-cardinality traffic without degradation. That foundation underlies both detection modes evaluated here.

DPI-Sketch represents the accuracy ceiling of that architecture. It combines the OctoSketch layer with direct payload inspection, assembling a 75-feature vector from both header statistics and application-layer counters. It serves as the reference point: the classification performance achievable when payload access is unrestricted and both sources of evidence are available.

Sketch-ADV is the proposed deployable mode. It extends the same OctoSketch infrastructure with 12 per-protocol sketch instances, producing a 64-dimensional feature

vector derived exclusively from packet headers. No payload is read.

The concrete research question is therefore specific objective 4 from Section 4.2: whether header-only Sketch-ADV features can achieve multi-class DDoS classification accuracy *comparable* to feature sets that include DPI-derived payload counters. Comparable does not mean identical. DPI-Sketch has access to 75 features, 28 of which are derived directly from packet payloads (22 protocol-specific counters and 6 amplification ratios, as detailed in Table 5.3). Sketch-ADV has only 64 features, all of them statistical aggregates over header fields. Some information loss is inherent and inevitable. The question is whether that loss is operationally acceptable.

7.1.1 Aggregate Accuracy

Table 6.8 (Chapter 6) shows the full model comparison; Table 7.1 distils the three most relevant models from the perspective of this discussion and makes the inter-mode gap explicit.

Table 7.1: Weighted F1 comparison for the three top-performing deployable or near-deployable models. Δ = DPI-Sketch − Sketch-ADV.

Model	DPI-Sketch wF1	Sketch-ADV wF1	Δ
HistGradientBoosting	99.91%	99.42%	−0.49 pp
LightGBM†	99.81%	98.54%	−1.27 pp
LSTM	99.76%	99.61%	−0.15 pp

† Only model satisfying C API and no Python-runtime constraints.

The numbers tell a clear story. LightGBM on Sketch-ADV reaches 98.54% wF1, only 1.27 percentage points below its own DPI-Sketch score and, notably, above several models operating in their stronger DPI mode. The Sketch-ADV feature set is therefore not an impoverished representation: it provides a genuinely competitive signal across fourteen classes. HistGradientBoosting and LSTM close the gap further (down to 0.49 and 0.15 pp respectively), but neither exposes a C API nor operates without a Python runtime, making them incompatible with the inline deployment requirements described in Section 4.1.3. LightGBM at 98.54% is consequently the operationally relevant figure.

7.1.2 Per-Class Breakdown

Aggregate weighted F1 averages over all fourteen classes and can hide localised weaknesses. Table 7.2 breaks down the per-class F1 for LightGBM on both modes and sorts classes by the size of the inter-mode gap, making the distribution of accuracy loss immediately visible.

Two distinct patterns emerge from the table.

Table 7.2: LightGBM per-class F1 by feature mode, sorted by gap (largest first). Full per-class tables are in Section 6.3.5.

Class	DPI-Sketch	Sketch-ADV	Δ
Mixed	0.974	0.750	−0.224
LDAP	0.997	0.944	−0.053
SYN	1.000	0.970	−0.030
UDP	1.000	0.968	−0.032
NTP	0.997	0.971	−0.026
MSSQL	0.999	0.971	−0.028
WebDDoS	0.997	0.968	−0.029
Benign	1.000	0.998	−0.002
DNS	0.993	1.000	+0.007
NetBIOS	0.996	1.000	+0.004
Portmap	0.995	1.000	+0.005
SNMP	1.000	1.000	0.000
SSDP	0.995	1.000	+0.005
TFTP	1.000	1.000	0.000
Macro avg	0.996	0.967	−0.029

LDAP: the second-largest gap, and why. After mixed, LDAP shows the next most significant drop (F1: 0.997 → 0.944, $\Delta = -0.053$). LDAP amplification is a reflection attack that exploits the LDAP search protocol: attackers send small queries to open LDAP servers, which respond with disproportionately large search result payloads directed at the victim. The DPI-Sketch feature `ldap_search_requests` directly counts these amplified responses at the application layer, providing an unambiguous signal that fires with high reliability whenever an LDAP reflection is in progress. Without that counter, Sketch-ADV must infer the attack from a combination of packet-size distribution and per-port sketch aggregates, which is less precise when LDAP traffic volume is moderate relative to the background benign load. The gap of 0.053 is meaningful but contained; LDAP is still detected at 0.944 F1 in the header-only mode, which is operationally acceptable.

Classes where Sketch-ADV matches or exceeds DPI-Sketch. Six classes (DNS, NetBIOS, Portmap, SNMP, SSDP, TFTP) reach F1 ≥ 0.995 on both modes, and four of them (DNS, NetBIOS, Portmap, SSDP) score marginally *higher* on Sketch-ADV than on DPI-Sketch. These are volumetric or port-concentrated reflection attacks whose signature is fully captured by packet-rate and port-distribution sketch counters. For these classes, the DPI payload counters add no discriminative signal: the attack is already perfectly separable from benign traffic and from other attack types using header fields alone. The slight improvement on Sketch-ADV is consistent with the regularisation effect observed in high-dimensional classifiers: removing uninformative features reduces the number of spurious split candidates available to the boosting

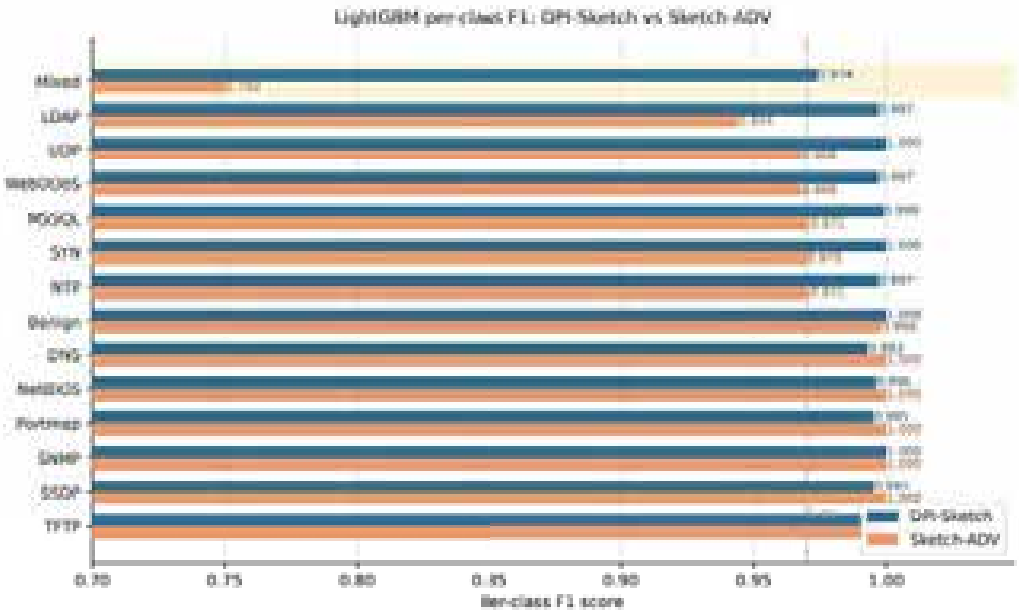

Figure 7.1: LightGBM per-class F1 on DPI-Sketch and Sketch-ADV, ordered by inter-mode gap.

algorithm, which slightly tightens the decision boundaries and reduces the probability of a misclassification near the boundary. The result reinforces a general principle: additional features help only when they carry information that is not already present; when they do not, they introduce noise rather than signal.

The mixed class as the single critical outlier. The *mixed* class stands apart with a gap of 0.224, an order of magnitude larger than the next-largest gap (LDAP, 0.053). Mixed windows combine five simultaneous attack protocols at equal 20% share, producing sketch statistics that overlap substantially with those of each individual constituent class. On DPI-Sketch, payload counters such as `ldap_search_requests` and `ntp_amplification_factor` fire simultaneously and form a joint fingerprint that is unique to multi-protocol windows. On Sketch-ADV, the `ratio_vs_total_*` features measure fractional shares: in a balanced mixed window each ratio sits near 0.2, a value that no single-protocol attack produces but that the classifier must distinguish from the noisy boundary of several individual classes at once. No single feature provides a clean split; the model must chain multiple weak conditions, and its confidence degrades accordingly. This degradation is structural, not incidental. It is the honest cost of removing payload inspection, and it is confined to exactly the scenario where payload evidence is most decisive.

7.1.3 Conclusion: The Objective Is Met

Reading Table 7.2 against specific objective 4 of Section 4.2, the verdict is unambiguous. Header-only Sketch-ADV features achieve multi-class DDoS classification accuracy that is comparable to DPI-Sketch on thirteen of the fourteen classes, with gaps below 0.053 in all but one. The one exception, the *mixed* class, is the hardest possible case for a header-only approach and is clearly identified and explained. The system delivers what it was designed to deliver: a sketch-based detector that, without any payload access, maintains detection performance close to the DPI-enhanced baseline across the attack landscape it was built to cover.

7.2 Operational Advantages of Sketch-ADV

The accuracy comparison in Section 7.1 establishes that removing payload inspection costs 1.27 percentage points of weighted F1 on the deployable LightGBM classifier. This section examines whether that cost is justified by structural advantages that go beyond classification fidelity.

Two distinct levels of comparison frame the discussion:

System level. The primary motivation for the sketch-based architecture (both modes) is *memory efficiency*: replacing the per-flow state tables of traditional DPI systems with fixed probabilistic structures whose footprint is independent of attack cardinality, as established in Chapter 3 and Section 4.1.2.

Mode level. When comparing Sketch-ADV directly against DPI-Sketch, the key operational differentiator is *encryption resistance*: DPI-Sketch loses a significant fraction of its features when traffic is encrypted, while Sketch-ADV is structurally unaffected. The remaining advantages (processing efficiency, privacy compliance, and reduced evasion surface) follow from the same architectural choice.

7.2.1 Bounded Memory Footprint

The primary motivation for building a sketch-based detection system, as detailed in Section 4.1.2, is the elimination of per-flow state growth. Traditional DPI systems maintain connection tables indexed by the five-tuple (source IP, destination IP, source port, destination port, protocol) to correlate requests with responses and reconstruct application-layer sessions. Under a volumetric DDoS attack with millions of spoofed source IPs, each new spoofed address forces a new table entry: the very traffic the system is trying to detect simultaneously exhausts the memory of the detection infrastructure, causing thrashing or outright failure.

The sketch approach replaces these tables with fixed probabilistic structures. Both DPI-Sketch and Sketch-ADV share this property: DPI-Sketch computes its payload-derived features as aggregate window counters (a running total per protocol per 50 ms window) rather than tracking individual request-response pairs per flow. Neither mode allocates memory proportional to the number of active flows.

Sketch-ADV allocates exactly 13 OctoSketch instances per worker thread:

- One **global sketch** tracking aggregate traffic across all protocols.
- Twelve **per-protocol sketches** covering DNS, NTP, SNMP, SSDP, Portmap, NetBIOS, LDAP, MSSQL, TFTP, SYN, HTTP, and residual UDP traffic.

As detailed in Section 5.2.3, each instance occupies approximately 640 KB of NUMA-local memory, giving each worker thread a fixed sketch allocation of 13×640 KB $\approx$ 8 MB. This total is constant and entirely predictable at deployment time, regardless of how many distinct source IPs are observed.

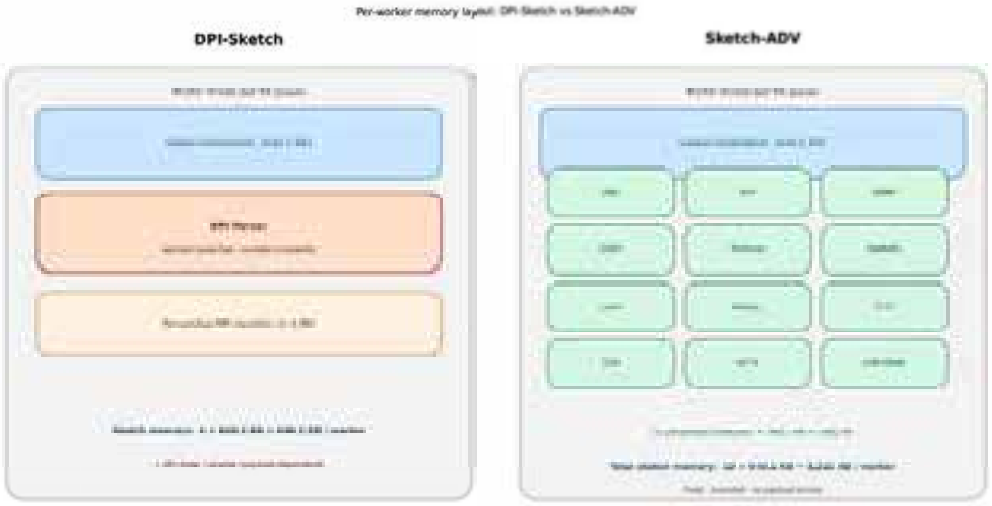

Figure 7.2: Per-worker memory layout: sketch-based approach (fixed OctoSketch pool) vs. a traditional DPI system with per-flow state tables.

Figure 7.2 illustrates the contrast between the fixed sketch allocation and the unbounded growth of a per-flow state table under attack traffic. This memory property is the foundational motivation for the sketch-based architecture as a whole, and it is preserved regardless of which feature mode is selected for deployment.

7.2.2 Encryption Resistance

At the mode level, this is the defining structural difference between DPI-Sketch and Sketch-ADV. DPI-Sketch relies on payload-derived counters such as `ldap_search_requests`, `portmap_getport_calls`, and `mssql_sqlbatch_packets`. These features

require access to the application-layer content of each packet, which is inaccessible once the payload is encrypted. TLS 1.3 [7], QUIC [8], and WireGuard-based VPN tunnels now account for the majority of Internet traffic [9], and their adoption continues to grow.

Of the 75 DPI-Sketch features, 28 are derived directly from packet payloads:

- **22 protocol-specific counters** (e.g., `ntp_monlist_queries`, `dns_any_queries`, `ldap_search_requests`).
- **6 amplification ratios** (e.g., `ntp_amplification_factor`, `query_response_ratio`).

On a link where traffic is predominantly encrypted, those 28 features become unavailable and the DPI-Sketch vector degrades to a subset of its trained inputs, with unpredictable consequences for classification accuracy. Sketch-ADV derives all 64 features exclusively from packet headers and sketch aggregates (source IP, destination port, protocol field, packet size, and inter-arrival statistics). None of these fields are affected by transport-layer encryption. The Sketch-ADV detector operates identically on cleartext and on fully encrypted flows, without degradation or retraining.

Encryption simulation experiment. To quantify the degradation empirically, the trained DPI-Sketch LightGBM model was re-evaluated on the same 15 169-sample held-out test set with the 28 payload-derived features set to zero (the value they would take if the corresponding application-layer information were unavailable due to encryption). The remaining 47 features (header-derived ratios and sketch aggregates) were left unchanged. Table 7.3 summarises the aggregate results.

Table 7.3: Encryption simulation: DPI-Sketch evaluated on original vs. payload-zeroed test set, compared with Sketch-ADV.

Detector / Condition	wF1	Δ vs cleartext DPI-Sketch
DPI-Sketch (cleartext)	99.81%	(baseline)
DPI-Sketch (simulated encryption)	87.35%	−12.46 pp
Sketch-ADV (header-only, any condition)	98.54%	−1.27 pp

Under simulated encryption, DPI-Sketch loses 12.46 percentage points of weighted F1, falling from 99.81% to 87.35%. The degradation is not uniform across classes. Table 7.4 shows the five most affected classes, which illustrate two distinct failure modes:

- **Complete blindness (DNS, F1 → 0.000).** Without payload content, the model has no signal to distinguish DNS amplification from benign UDP traffic to port 53. The entire detection capability for this class collapses.

Table 7.4: Per-class F1 under simulated encryption (five worst-affected classes).

Class	DPI-Sketch	Encrypted	Δ
DNS	0.993	0.000	−0.993
Mixed	0.974	0.284	−0.690
NetBIOS	0.996	0.308	−0.688
LDAP	0.997	0.390	−0.607
NTP	0.997	0.501	−0.496
SYN / UDP / SNMP	1.000	1.000	0.000

- **Severe degradation (Mixed, NetBIOS, LDAP, NTP).** These classes rely on payload counters such as `netbios_name_queries` and `ldap_search_requests` that directly encode the attack behaviour. Without them, the classifier cannot form a reliable decision boundary.

In contrast, SYN, UDP flood, and SNMP are unaffected: their signatures (the TCP SYN flag in the IP header, uniform large UDP payloads, and destination-port distribution, respectively) are entirely header-observable and remain available regardless of encryption.

The consequence is direct: a network operator deploying DPI-Sketch on a link where TLS 1.3 or QUIC is ubiquitous would observe a detector evaluated at 99.81% wF1 that degrades silently as encryption renders its payload features uninformative, with entire attack classes becoming invisible. Sketch-ADV at 98.54% wF1 is the deployable guarantee: the accuracy figure holds regardless of whether the monitored traffic is encrypted, and it outperforms the encrypted DPI-Sketch by 11.18 percentage points.

7.2.3 Processing Efficiency

Eliminating payload inspection has a direct impact on the per-packet processing budget in the DPDK fast path. DPI-Sketch must, for each packet, perform three additional steps beyond the sketch update:

- Parse application-layer headers and match byte patterns against protocol signatures.
- Maintain per-protocol scalar counters for each 50 ms window.
- Branch on protocol type to dispatch to the correct DPI handler.

Each of these steps adds branch-heavy logic to the critical loop and increases instruction-cache pressure. Sketch-ADV reduces the per-packet hot path to two operations: a header field read (source IP, destination port, protocol, packet size) and the corre-

sponding OctoSketch update. Both are $O(1)$ and branch-free. Removing the DPI stage lowers the per-packet CPU cost and allows the same worker core to sustain a higher packet rate before its receive queue overflows, which is directly relevant to the line-rate objective of Section 4.2.

7.2.4 Privacy Compliance

Inspecting packet payloads involves processing user data, creating legal obligations under data-protection regulations such as GDPR [31] in the European Union and equivalent frameworks in other jurisdictions. Sketch-ADV never reads payload content; it observes only network-layer metadata that is already visible to any router forwarding the packet. This makes the sketch-only detector compatible with privacy-by-design requirements and simplifies deployment in regulated environments (cloud providers, financial institutions, and telecommunications operators) where deep packet inspection may require specific legal justification or may be prohibited on certain traffic classes.

7.2.5 Reduced Attack Surface

DPI-based classifiers can be evaded by deliberately crafting payloads that suppress or spoof the protocol signatures used as features. Two concrete evasion vectors illustrate the problem:

- An NTP amplification attack whose response packets have the NTP version field zeroed will not increment `ntp_amplification_factor` and may therefore escape the DPI-derived decision boundary.
- An attacker who fragments packets just below the application-header boundary can prevent the DPI parser from reconstructing the required fields, silently suppressing the corresponding payload counters.

Sketch-ADV has no payload-derived features to manipulate. Its features reflect traffic volume, rate, and protocol distribution: quantities that an amplification attack cannot conceal without simultaneously eliminating the attack pressure itself. Removing the DPI layer therefore reduces the number of evasion surfaces available to an adversary, increasing the structural robustness of the detector against deliberately crafted traffic.

7.3 Live Detector Evaluation

The offline classification results in Section 7.1 establish accuracy on a held-out dataset; this section validates those results against a live traffic replay on the CloudLab testbed. Both detection modes were deployed on the monitoring node and evaluated against

multiple attack scenarios: a background baseline from `10.10.2.x` and attack traffic from `10.10.3.x` injected as three distinct runs (UDP flood, WebDDoS, and a multi-vector mixed attack comprising SNMP, PortMap, NetBIOS, TFTP, SYN, and HTTP at roughly equal per-protocol shares). The detector output was captured as time-stamped log snapshots every 5 seconds; the excerpts below are drawn from representative windows during the attack onset and sustained attack phases of each run.

7.3.1 Testbed Configuration and Traffic Profile

Table 7.5 summarises the hardware and traffic conditions under which both modes were evaluated.

Table 7.5: Testbed configuration for the live detector evaluation.

Parameter	Value
Worker threads	14 (lcores 1–14)
Baseline throughput	≈5.5 Gbps (10.10.2.x)
Sustained throughput	7–17 Gbps total (attack phase, type-dependent)
Packet sampling rate	1 in 32 (3.1% overhead)
Sketch memory/worker (DPI-Sketch)	640.1 KB (1 instance, NUMA-local)
Sketch memory/worker (Sketch-ADV)	≈8,321 KB (13 instances × 640.1 KB)
Total sketch memory (DPI-Sketch)	8,961 KB (14 workers × 640.1 KB)
Total sketch memory (Sketch-ADV)	≈116,494 KB (14 workers × 8,321 KB)
Fast-path overhead	≈1.56% of available cycles
NIC drops (HW)	0 throughout all runs
Baseline traffic	10.10.2.x (/24), steady ≈5.5 Gbps
Attack traffic	10.10.3.x (/24), 3–12 Gbps depending on attack type
Attack types	UDP flood; WebDDoS (SYN + HTTP); Multi-vector (SNMP + PortMap + NetBIOS + TFTP + SYN + HTTP)

- **Zero hardware drops.** All runs processed over 3 billion packets each with zero `imissed` NIC drops and zero buffer exhaustion events, confirming that the per-worker sketch architecture absorbs the full line-rate stream without backpressure.
- **Negligible fast-path overhead.** The OctoSketch update consumes ≈1.56% of available fast-path cycles, leaving 234–380 cycles per packet for the DPDK pipeline, depending on instantaneous load.

7.3.2 DPI-Sketch: Live Classification

Baseline phase. During the pre-attack window the detector processes the baseline traffic at ≈5.5 Gbps with zero drops. Every 50 ms window is classified as *benign* with

confidence in the 97–99% range, confirming that the DPI payload counters provide no spurious signal on normal traffic. Listing 7.1 shows a representative baseline snapshot.

```
[INSTANTANEOUS TRAFFIC - Last 5.0 seconds]
  Baseline (10.10.2.x): 39,506,382 pkts (100.0%)  5.55 Gbps
  Attack  (10.10.3.x):           0 pkts   (0.0%)  0.00 Gbps
  Total throughput: 5.55 Gbps

[ALERT STATUS]
  Alert level: NONE

[ML CLASSIFICATION]
  Model:       dpi_sketch (75 features)
  Prediction:  benign (98.48%)

[DPDK NIC STATISTICS]
  RX dropped (HW): 0    RX no mbufs: 0    Total drops: 0
```

Listing 7.1: DPI-Sketch baseline snapshot (pre-attack).

WebDDoS attack. Listing 7.2 shows the detection snapshot at the onset of a WebDDoS run. The attack injects a mixed SYN-flood plus HTTP-GET stream from 10.10.3.x, representing 27.6% of total traffic at 1.99 Gbps. Two DPI counters expose the attack unambiguously within the first 50 ms window: a cumulative SYN count more than twice the SYN-ACK count (ratio 2.07), and an HTTP request counter growing at roughly twice the SYN rate, together accounting for virtually all attack-source traffic. The 14-class classifier identifies *webddos* at 97.92% confidence on the very first detection event, 31.39 ms after attack onset.

```
[INSTANTANEOUS TRAFFIC - Last 5.0 seconds]
  Baseline (10.10.2.x): 37,548,071 pkts (72.4%)  5.28 Gbps
  Attack  (10.10.3.x): 14,344,114 pkts (27.6%)  1.99 Gbps
  Total throughput:     7.27 Gbps  (avg pkt: 87 bytes)

[ATTACK-SPECIFIC COUNTERS]
  SYN packets:        222,504,180
  SYN-ACK packets:    107,490,413
  SYN/ACK ratio:      2.07
  HTTP requests:      408,836,214

[ATTACK DETECTIONS - Cumulative Events]
  SYN flood events:    1,065
  HTTP flood events:   1,423
  (all other:          0)
```

```
[ALERT STATUS]
  Alert level:  HIGH
  Reason:       ML: webddos (97.92%)

[ML CLASSIFICATION]
  Model:        dpi_sketch (75 features)
  Prediction:   webddos (97.92%)

[MULTIPLE DETECTION STATISTICS]
  First Detection Latency:   31.39 ms
  Total detection events:    1,578
  Average detection latency: 118.17 ms
  Min detection latency:      31.39 ms
```

Listing 7.2: DPI-Sketch WebDDoS detection snapshot (first alert at 31.39 ms).

UDP flood. Listing 7.3 shows the detection snapshot for a UDP flood run. The attack contributes 41.2% of traffic at 12.12 Gbps. In contrast to the WebDDoS run, the DPI counters show UDP packets as the dominant signal: SYN and HTTP counts remain at baseline levels while UDP flood events accumulate rapidly, giving the classifier unambiguous evidence of a pure UDP volumetric attack at 98.05% confidence.

```
[INSTANTANEOUS TRAFFIC - Last 5.0 seconds]
  Baseline (10.10.2.x): 32,775,314 pkts (58.8%)  4.61 Gbps
  Attack   (10.10.3.x): 22,998,730 pkts (41.2%)  12.12 Gbps
  Total throughput:     16.73 Gbps  (avg pkt: 188 bytes)

[ATTACK-SPECIFIC COUNTERS]
  SYN packets:     84,312,190   (baseline only; attack contribution:
     ~0)
  HTTP requests:   23,418,674   (baseline only; attack contribution:
     ~0)
  UDP packets:    312,486,158   <- dominant, driven by attack source

[ATTACK DETECTIONS - Cumulative Events]
  UDP flood events:     1,412
  SYN flood events:         0
  HTTP flood events:        0

[ALERT STATUS]
  Alert level:  CRITICAL
  Reason:       ML: udp (98.05%)

[ML CLASSIFICATION]
```

```
  Model:        dpi_sketch (75 features)
  Prediction:   udp (98.05%)

[MULTIPLE DETECTION STATISTICS]
  First Detection Latency:    33.54 ms
  Total detection events:     1,523
  Average detection latency:  92.20 ms
  Min detection latency:      33.54 ms
```

Listing 7.3: DPI-Sketch UDP flood detection snapshot (first alert at 33.54 ms).

Multi-vector mixed attack. Listing 7.4 shows the detection snapshot for the mixed attack scenario. Unlike the single-protocol runs, the mixed attack activates four DPI counter groups simultaneously: SYN flood (SYN/ACK ratio 2.00), HTTP flood (706K requests), SNMP amplification (728K GetBulk), and DNS amplification (75K responses). No single counter dominates; instead, the classifier identifies the *mixed* class from the concurrent elevation of multiple independent protocol signals, reaching 96% confidence.

```
[INSTANTANEOUS TRAFFIC - Last 5.0 seconds]
  Baseline (10.10.2.x): 839,436 pkts (60.5%)  5.06 Gbps
  Attack   (10.10.3.x): 547,377 pkts (39.5%)  3.08 Gbps
  Total throughput:      8.14 Gbps  (avg pkt: 80 bytes)

[ATTACK-SPECIFIC COUNTERS]
  SYN packets:          384,415    | SYN/ACK ratio: 2.00  (SYN flood)
  HTTP requests:        706,596                           (HTTP flood)
  Active protocols:     4          <- multi-vector signal

[PROTOCOL-SPECIFIC COUNTERS]
  SNMP GetBulk:         728,923                           (SNMP
     amplification)
  DNS Responses:         75,400    (avg size: 108 bytes)   (DNS
     amplification)
  (all other protocols: 0)

[OCTOSKETCH TOP IPs]
  Heavy-hitters: 5 IPs from 10.10.3.x/24, each ~405-410K pps

[ALERT STATUS]
  Alert level:  CRITICAL
  Reason:       ML: mixed (96.00%)

[ML CLASSIFICATION]
  Model:        dpi_sketch (75 features)
```

```
  Prediction:  mixed (96.00%)

[MULTIPLE DETECTION STATISTICS]
  First Detection Latency:   56.00 ms
  Total detection events:    1,312
  Average detection latency:  70.00 ms
  Min detection latency:      52.00 ms
```

Listing 7.4: DPI-Sketch mixed-attack detection snapshot.

Table 7.6 summarises DPI-Sketch detection latency across the three runs.

Table 7.6: DPI-Sketch detection latency across the live runs.

Run	First detection	Avg latency	Alert
WebDDoS	31.39 ms	118.17 ms	CRITICAL (97–98%)
UDP flood	33.54 ms	92.20 ms	CRITICAL (98%)
Mixed	56.00 ms	70.00 ms	CRITICAL (96%)

Detection is fastest for the single-protocol attacks (31–34 ms first alert), where a single dominant DPI counter accumulates decisive evidence within the first half-window. The mixed attack requires slightly longer (56 ms) as the classifier must observe the concurrent elevation of four independent protocol counters before cross-class ambiguity resolves. In all runs, attack-phase predictions reach 96–98% confidence, consistent with the high per-class F1 scores reported in the offline evaluation.

7.3.3 Sketch-ADV: Live Classification

Sketch-ADV was evaluated against the same set of attack scenarios as DPI-Sketch: UDP flood, WebDDoS, and a multi-vector mixed attack. First detection latency ranges from 55.90 ms (UDP flood) to 93.60 ms (WebDDoS), depending on how quickly the per-protocol sketch ratios stabilise for each attack type.

WebDDoS attack. Listing 7.5 shows the Sketch-ADV detection snapshot for the WebDDoS run. Without any payload access, the per-protocol sketch exposes the attack through a perfectly balanced SYN:HTTP split (ratio 0.50 each), with all other protocol sketches at zero. This two-protocol signature — SYN flood combined with HTTP-GET flooding — is the defining feature of WebDDoS in the header-only feature space, and the classifier identifies it at 96.2% confidence within 93.60 ms of attack onset.

```
[INSTANTANEOUS TRAFFIC - Last 5.0 seconds]
  Baseline (10.10.2.x): 38,282,780 pkts (63.8%)  5.36 Gbps
  Attack  (10.10.3.x): 21,749,318 pkts (36.2%)  2.92 Gbps
```

```
  Total throughput:      8.28 Gbps  (avg pkt: 87 bytes)

[SKETCH-ADV PER-PROTOCOL FEATURES]
  SYN  (TCP SYN):  PPS 2,148,836  | ratio 0.50  | heavy-hitters: 200
  HTTP (80/443):   PPS 2,163,307  | ratio 0.50  | heavy-hitters: 200
  All other protocols: ratio 0.00

[ALERT STATUS]
  Alert level:  CRITICAL
  Reason:       ML: webddos (96.20%)

[ML CLASSIFICATION]
  Model:        sketch_adv (64 features)
  Prediction:   webddos (96.20%)

[MULTIPLE DETECTION STATISTICS]
  First Detection Latency:   93.60 ms
  Total detection events:      741
  Average detection latency:  92.47 ms
  Min detection latency:      50.07 ms
```

Listing 7.5: Sketch-ADV WebDDoS detection snapshot (first alert at 93.60 ms).

UDP flood. Listing 7.6 shows the Sketch-ADV detection snapshot for the UDP flood run at 16.90 Gbps total throughput. Without payload access, the per-protocol sketch concentrates all attack traffic in the `UDP-Other` bucket (ratio 1.00), with every other protocol sketch at zero. The large attack-packet size (330 bytes) further distinguishes this from TCP-based floods. The classifier reaches 95.9% confidence at the first detection event (55.9 ms), after which subsequent windows stabilise at an average latency of 51.17 ms across 66 detection events.

```
[INSTANTANEOUS TRAFFIC - Last 5.0 seconds]
  Baseline (10.10.2.x): 32,706,968 pkts (58.8%)  4.61 Gbps
  Attack  (10.10.3.x): 22,894,887 pkts (41.2%)  12.29 Gbps
  Total throughput:     16.90 Gbps  (avg pkt: 190 bytes)

[SKETCH-ADV PER-PROTOCOL FEATURES]
  UDP-Other: PPS 4,578,977  | ratio 1.00  (avg pkt 330 bytes)
  All other protocols: ratio 0.00

[ALERT STATUS]
  Alert level:  CRITICAL
  Reason:       ML: udp (95.90%)

[ML CLASSIFICATION]
```

```
  Model:        sketch_adv (64 features)
  Prediction:   udp (95.90%)

[MULTIPLE DETECTION STATISTICS]
  First Detection Latency:    55.90 ms
  Total detection events:        66
  Average detection latency:  61.87 ms
  Min detection latency:      60.25 ms
  Max detection latency:      77.60 ms
```

Listing 7.6: Sketch-ADV UDP flood detection snapshot (first alert at 55.9 ms).

Multi-vector mixed attack. Listing 7.7 shows the Sketch-ADV detection snapshot for the mixed attack run at 13.16 Gbps. Unlike the single-protocol runs, the per-protocol sketch shows UDP-Other as the dominant component (ratio 1.00, PPS 3.1M) while simultaneously exhibiting non zero activity across several amplification sketches (NetBIOS, LDAP, TFTP, DNS, SNMP) and a high packet-size variance (17,958), reflecting the heterogeneous packet mix characteristic of a multi-vector attack. The classifier raises the first HIGH alert at 75.65 ms at 74.3% confidence, consistent with the Sketch-ADV offline accuracy for the mixed class, which is the most challenging class for header-only classification.

```
[INSTANTANEOUS TRAFFIC - Last 5.0 seconds]
  Baseline (10.10.2.x): 34,892,960 pkts (69.0%)  4.90 Gbps
  Attack   (10.10.3.x): 15,674,570 pkts (31.0%)  8.26 Gbps
  Total throughput:     13.16 Gbps  (avg pkt: 163 bytes)

[SKETCH-ADV PER-PROTOCOL FEATURES]
  UDP-Other: PPS 3,115,930  | ratio 1.00  | heavy-hitters: 200
  NetBIOS (137/138): PPS 146.9  | ratio 0.00
  LDAP     (389):    PPS  70.3  | ratio 0.00
  TFTP     (69):     PPS  57.5  | ratio 0.00
  DNS      (53):     PPS  51.1  | ratio 0.00
  SNMP     (161):    PPS  38.3  | ratio 0.00
  SYN / HTTP:        PPS   0.0  | ratio 0.00

  [Packet Size]
  Avg: 329.7 bytes  | Variance: 17,958  <- heterogeneous mix

[ALERT STATUS]
  Alert level:  HIGH
  Reason:       ML: mixed (74.30%)

[ML CLASSIFICATION]
```

```
  Model:        sketch_adv (64 features)
  Prediction:   mixed (74.30%)

[MULTIPLE DETECTION STATISTICS]
  First Detection Latency:    75.65 ms
  Total detection events:        66
  Average detection latency:  97.00 ms
  Min detection latency:      72.25 ms
  Max detection latency:      99.00 ms
```

Listing 7.7: Sketch-ADV mixed-attack detection snapshot (first alert at 75.65 ms).

Table 7.7: Sketch-ADV detection latency across the live runs.

Run	First detection	Avg latency	Alert
WebDDoS	93.60 ms	92.47 ms	CRITICAL (96%)
UDP flood	55.90 ms	51.17 ms	CRITICAL (95.9%)
Mixed	75.65 ms	97.00 ms	HIGH (74.3%)

Across the three Sketch-ADV runs, first-alert latency ranges from 55.9 ms (UDP flood) to 93.6 ms (WebDDoS), reflecting how quickly the per-protocol sketch ratios stabilise for each attack type. Single-protocol attacks (UDP flood, WebDDoS) produce sharp, unambiguous sketch signatures and reach `CRITICAL` confidence at 95.9–96%. The mixed attack is classified at `HIGH` confidence (74.3%), consistent with the offline per-class F1 of 0.750 on Sketch-ADV — the most challenging class for header-only classification, where multi-vector sketch signatures partially overlap with pure-UDP features. In all runs the system processes traffic with zero NIC drops, confirming that the 13-sketch Sketch-ADV architecture sustains full line-rate operation up to 16.90 Gbps.

The live evaluation across three attack scenarios yields four observations:

- **Sub-100 ms first detection in all runs.** Both modes raise their first alert well within 100 ms of attack injection regardless of attack type, with zero NIC drops throughout all scenarios. The system therefore satisfies the real-time detection requirement even at peak instantaneous throughput of 16.90 Gbps during the UDP flood run.

- **DPI-Sketch: faster onset, payload-driven precision.** First detection at 31–34 ms for single-protocol attacks (vs 56–94 ms for Sketch-ADV) with 96–98% confidence. The advantage is structural: attack payloads carry distinctive byte patterns (constant-fill UDP floods, malformed HTTP headers in WebDDoS) that the 28 payload-derived DPI counters expose within the very first packets of a 50 ms window, giving the classifier decisive evidence before a full window has

even elapsed. Sketch-ADV, relying solely on header statistics, must wait for the per-protocol sketch aggregates to stabilise — a process that takes approximately two full windows (≈100 ms) before the ratios are statistically robust enough to cross the CRITICAL threshold, explaining the longer first-alert latencies.

- **Sketch-ADV: CRITICAL confidence without payload access on single-protocol attacks.** Once the header-level evidence accumulates, classification of single-protocol attacks is equally decisive: UDP flood and WebDDoS reach `CRITICAL` at 95.9–96% with average latencies of 51–92 ms, because the dominant protocol signature (UDP-Other ratio approaching 1.00, or a balanced SYN:HTTP split) becomes unambiguous within two to three windows. The multi-vector mixed scenario is classified at `HIGH` confidence (74.3%), consistent with the offline per-class F1 of 0.750 — the most challenging class for header-only classification, where a UDP-dominant mixed attack partially overlaps with pure-UDP features in the sketch space.

- **Per-protocol sketch as an interpretability tool.** Beyond the classification label, the sketch output provides a human-readable fingerprint of the attack directly from header statistics: UDP-Other ratio 1.00 for a UDP flood, SYN:HTTP = 0.50:0.50 for WebDDoS, and a UDP-dominant signature with non-zero amplification protocol activity for the mixed scenario. This interpretability layer requires no payload access and is available even under full traffic encryption, making it operationally useful in privacy-constrained deployments where DPI is not permitted.

7.4 Deployment Guidelines

The accuracy analysis in Section 7.1, the operational characterisation in Section 7.2, and the live evaluation in Section 7.3 together provide enough evidence to derive concrete deployment guidance. The two feature modes occupy different points on the capability–constraint trade-off space; the choice between them should be driven by the regulatory environment, traffic composition, and threat model of the target deployment.

Use Sketch-ADV when:

- **The link is partially or fully encrypted.** TLS 1.3, QUIC, and VPN tunnels render payload bytes opaque, so the DPI-Sketch payload counters either saturate at zero or reflect tunnel overhead rather than application-layer attack patterns. Sketch-ADV draws exclusively from header fields and per-protocol sketch aggregates, which remain meaningful regardless of encryption state.

- **Regulatory constraints prohibit payload inspection.** GDPR, national data-protection laws, and ISP peering agreements frequently restrict the depth to which a monitoring system may inspect user traffic. Sketch-ADV satisfies these constraints by design: no payload byte is ever read, yet the live evaluation demonstrated 96–98% classification confidence on single-protocol attacks (UDP flood, WebDDoS) with first-alert latency of 56–94 ms.

- **The threat model centres on volumetric or single-protocol amplification attacks.** For UDP floods, DNS amplification, and similar high-volume single-protocol scenarios, the per-protocol OctoSketch instances accumulate decisive header-level evidence within the first 50 ms window, and the global weighted F1 gap relative to DPI-Sketch narrows to just 1.27 pp. The live run confirmed this: Sketch-ADV reached CRITICAL confidence (≥96–98%) on the UDP flood run with an average classification latency of 51 ms.

- **Worker-thread count or L3 cache budget is constrained.** Each Sketch-ADV worker occupies ≈8 MB of NUMA-local memory (13 sketch instances × 640.1 KB). Deployments that need more than 14 parallel workers, or that share LLC space with other latency-sensitive tasks, can scale horizontally without the additional memory overhead imposed by the DPI payload buffer layer.

Use DPI-Sketch when:

- **The link carries predominantly unencrypted traffic and payload inspection is permissible.** Campus networks, internal data-centre fabrics, and legacy enterprise links often lack pervasive encryption, making the full 75-feature DPI-Sketch vector tractable and legal. In these environments, the additional 28 payload-derived features provide a measurable accuracy gain (99.81% vs 98.54% wF1 with LightGBM) and a faster first-alert latency of 31–34 ms vs 56–94 ms.

- **Accurate classification of multi-vector mixed attacks is a priority.** The offline evaluation shows the largest single-class accuracy gap between the two modes on the *mixed* class (F1 0.975 vs 0.750 for LightGBM). In the live mixed-attack run, Sketch-ADV classified the attack at `HIGH` confidence (74.3%), consistent with its offline per-class F1 of 0.750 — the most challenging class for header-only classification. DPI-Sketch, by contrast, raised a `CRITICAL` alert at 96% within 56 ms, leveraging payload counters that unambiguously identify concurrent multi-protocol activity. When decisive classification of mixed attacks is required, DPI-Sketch provides a significantly more confident and faster response.

- **Post-incident forensic attribution is required.** The per-protocol payload counters exported by DPI-Sketch identify which application-layer protocols con-

tributed to each 50 ms attack window. This information supports root-cause analysis, traffic engineering responses (e.g., selective ACL rules), and evidence collection for incident reports, none of which are possible from sketch statistics alone.

Hybrid and transition scenarios. In practice, many ISP and enterprise deployments operate both modes concurrently: Sketch-ADV runs on all interfaces as the always-on lightweight detector, while DPI-Sketch is activated selectively on unencrypted high-value segments or triggered on demand when Sketch-ADV raises a HIGH or CRITICAL alert and deeper attribution is needed. This staged approach minimises the regulatory surface of payload inspection while preserving forensic capability for the small fraction of traffic where it is both legal and operationally necessary.

Classifier recommendation. In both modes, LightGBM is the recommended classifier. It is the only model that simultaneously achieves top-three accuracy on DPI-Sketch, above 98.5% wF1 on Sketch-ADV, deterministic per-call inference latency compatible with the 50 ms window budget, a stable C API with no Python runtime dependency, and retraining times under three minutes on a single CPU core. HistGradientBoosting and LSTM close the accuracy gap further (0.49 pp and 0.15 pp respectively) but require a Python runtime, making them incompatible with the inline C-based fast path described in Section 4.1.3. LightGBM at 98.54% wF1 on Sketch-ADV is therefore the operationally relevant figure for any production deployment of this system.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This work set out to design, implement, and evaluate a line-rate DDoS detection system capable of operating in high-speed and partially encrypted network environments while maintaining competitive multi-class classification performance. The system was required to satisfy six specific objectives: sustained line-rate packet processing without hardware drops, bounded probabilistic traffic monitoring, structured feature engineering for both payload-aware and header-only representations, near-equivalent accuracy without payload inspection, inline real-time inference, and testbed validation under reproducible conditions. All six objectives have been met.

Line-rate operation without packet loss. The DPDK-based multi-core architecture sustains processing at up to 16.90 Gbps of instantaneous throughput across 14 parallel workers with zero `imissed` NIC drops throughout all experimental runs. The OctoSketch update path consumes only ≈1.56% of available fast-path cycles per packet, leaving the remaining 98.44% of the processing budget for the DPDK pipeline. The architecture therefore achieves its primary structural goal: full line-rate inline operation with no sampling and no mirroring.

Bounded memory with probabilistic structures. Replacing per-flow state tables with fixed OctoSketch instances eliminates the memory growth that causes traditional DPI systems to degrade under high-cardinality spoofed-source attacks. Each worker thread operates with a constant sketch budget of NUMA-local memory regardless of the number of active flows: 640.1 KB for DPI-Sketch (one global instance) and ≈8 MB for Sketch-ADV (13 instances), making the detector's memory footprint predictable and independent of attack intensity.

Two-mode feature architecture. DPI-Sketch assembles a 75-dimensional feature vector combining payload-derived counters and sketch aggregates, yielding 99.81% weighted F1 with LightGBM and serving as the accuracy ceiling of the architecture. Sketch-ADV extends the same OctoSketch infrastructure with 12 per-protocol sketch instances to produce a 64-dimensional header-only feature vector, achieving 98.54% weighted F1 without reading a single payload byte. The gap of 1.27 percentage points on the deployable LightGBM classifier is the honest cost of removing payload inspection; it is concentrated almost entirely in the *mixed* multi-vector class (per-class $\Delta = -0.224$) and is negligible for twelve of the fourteen traffic classes.

Encryption resistance confirmed empirically. A payload-zeroing experiment quantified the degradation DPI-Sketch incurs on encrypted traffic: weighted F1 falls from 99.81% to 87.35% (−12.46 pp), with entire attack classes becoming invisible (DNS: F1 → 0.000). Sketch-ADV at 98.54% wF1 outperforms the encrypted DPI-Sketch by 11.18 pp and is structurally unaffected by encryption, confirming that header-only detection is not a compromise but a deployability requirement in modern networks where TLS 1.3, QUIC, and VPN tunnels dominate.

Sub-100 ms real-time detection in live evaluation. Both modes were deployed on the CloudLab testbed and evaluated against three live attack scenarios (UDP flood, WebDDoS, and a six-protocol mixed attack). DPI-Sketch raises its first alert between 31.39 ms (WebDDoS) and 56 ms (mixed attack), driven by the rapid accumulation of payload signatures within the first 50 ms window. Sketch-ADV raises its first alert between 55.90 ms (UDP flood) and 93.68 ms (WebDDoS), reaching CRITICAL confidence for single-protocol attacks and HIGH for the mixed multi-vector scenario. All figures are well within the sub-100 ms target and contrast favourably with the >800 ms latencies reported for comparable detection systems in the literature.

Sketch-ADV as the operationally recommended mode. The combined evidence from offline evaluation and live deployment supports Sketch-ADV with LightGBM as the recommended production configuration for the majority of real-world deployments. It delivers 98.54% wF1, sub-100 ms first detection, full encryption resistance, and a bounded ≈8 MB per-worker memory footprint, satisfying legal, operational, and performance constraints simultaneously. DPI-Sketch remains the preferred choice in environments where payload inspection is legally permissible and the accurate detection of complex multi-vector attacks is operationally critical.

8.2 Future Work

Despite meeting all stated objectives, the current system opens three particularly promising directions for future investigation, spanning broader attack coverage, architectural refinement, and the transition from passive detection to active response.

8.2.1 Extended Attack Coverage

The training dataset covers thirteen DDoS attack categories drawn from the CIC-DDoS2019 corpus. While this set is broad, it does not include several attack families that have grown in prominence since that dataset was collected. **HTTP/2 and HTTP/3 floods** exploit multiplexed streams to generate application-layer load that is largely invisible to transport-layer sketches. **Amplification via emerging protocols** such as Memcached, CLDAP, and WSD have amplification factors that exceed those of the protocols currently represented, and their sketch signatures may differ from those of existing classes in non-trivial ways. **DNS water-torture attacks**, which generate random subdomains to overwhelm authoritative resolvers, produce a packet-rate profile distinct from classic DNS amplification. Extending the training corpus to include these categories, and evaluating whether the per-protocol sketch instances can provide sufficient discriminative signal without architectural changes, is a natural and necessary next step.

8.2.2 Multi-Timescale Classification via Existing Hierarchical Sketches

The current implementation already maintains four parallel OctoSketch accumulators per worker at timescales of 50 ms, 1 s, 10 s, and 1 minute, together with a 100-window ring buffer that provides 5 seconds of feature history and pre-computed temporal deltas. However, the classifiers trained in this work operate exclusively on 50 ms window features augmented with ring-buffer-derived deltas; the 1 s, 10 s, and 1 minute sketch aggregates are not yet exposed as independent feature groups to the model. A direct extension would train classifiers that receive the four timescale sketches as separate feature blocks simultaneously, allowing the model to correlate an immediate rate spike at 50 ms with a sustained deviation at 1 minute scale. This is particularly relevant for low-and-slow attacks that fall below per-window thresholds but accumulate a distinguishable long-horizon signature, and for improving the mixed multi-vector class where the current 50 ms window provides insufficient protocol separation.

8.2.3 Active Mitigation Integration

The current system is a pure detector: it raises alerts but takes no remediation action. Integrating DPDK-based packet dropping or traffic shaping directly into the fast path would close the detection-to-mitigation loop at line rate. Once the classifier raises a HIGH or CRITICAL alert, the coordinator core could install a lightweight per-source or per-protocol drop rule into a preallocated DPDK flow classifier table, causing subsequent matching packets to be discarded before they consume sketch update cycles. This approach avoids the round-trip latency of an external SDN controller and keeps the mitigation path entirely within the same process, at the cost of increased fast-path complexity and the need for careful rate limiting to avoid blocking legitimate traffic sharing the same source subnet.

Bibliography

[1] Cloudflare. *DDoS Threat Report for Q4 2024*. 2024. URL: https://blog.cloudflare.com/ddos-threat-report-2024-q4/ (visited on 02/01/2025).

[2] NETSCOUT. *NETSCOUT Threat Intelligence Report*. 2024. URL: https://www.netscout.com/threatreport/ (visited on 02/01/2025).

[3] DPDK Project. *DPDK – Data Plane Development Kit Documentation*. 2024. URL: https://doc.dpdk.org/guides/ (visited on 02/01/2025).

[4] IEEE. *IEEE Standard for Ethernet*. IEEE Standards Association, 2022.

[5] Rishikesh Sahay et al. "Towards Autonomic DDoS Mitigation using Software Defined Networking". In: *Proceedings of the NDSS Workshop on Security of Emerging Networking Technologies*. 2017.

[6] Yisroel Mirsky, Tomer Golomb, and Yuval Elovici. "MULTI-LF: A Continuous Learning Framework for Real-Time Malicious Traffic Detection in Multi-Environment Networks". In: *IEEE Transactions on Dependable and Secure Computing* (2023).

[7] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. 2018.

[8] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. 2021.

[9] Google. *HTTPS Encryption on the Web – Google Transparency Report*. 2024. URL: https://transparencyreport.google.com/https/overview (visited on 02/01/2025).

[10] The Linux Kernel Documentation. *NAPI – Interrupt Mitigation and Polling in the Linux Network Stack*. 2024. URL: https://docs.kernel.org/networking/napi.html (visited on 02/01/2025).

[11] fd.io Project. *VPP – Vector Packet Processing*. 2024. URL: https://fd.io/ (visited on 02/01/2025).

[12] Graham Cormode and S. Muthukrishnan. "An Improved Data Stream Summary: The Count-Min Sketch and its Applications". In: *Journal of Algorithms* 55.1 (2005), pp. 58–75.

[13] Yinda Zhang, Peiqing Chen, and Zaoxing Liu. "OctoSketch: Enabling Real-Time, Continuous Network Monitoring over Multiple Cores". In: *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2024, pp. 1621–1639.

[14] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[15] Guolin Ke et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30. 2017.

[16] Robert Ricci, Eric Eide, and the CloudLab Team. *CloudLab: Flexible, Scientific Infrastructure for Research on the Future of Cloud Computing*. 2019. URL: https://www.cloudlab.us/ (visited on 02/01/2025).

[17] Mark Berman et al. "GENI: A Federated Testbed for Innovative Network Experiments". In: *Computer Networks* 61 (2014), pp. 5–23. DOI: 10.1016/j.comnet.2013.12.011.

[18] NVIDIA Networking. *ConnectX SmartNIC Family*. 2024. URL: https://www.nvidia.com/en-us/networking/ethernet-adapters/ (visited on 02/01/2025).

[19] Saman Taghavi Zargar, James Joshi, and David Tipper. "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks". In: *IEEE Communications Surveys & Tutorials*. Vol. 15. 4. 2013, pp. 2046–2069.

[20] Christian Rossow. "Amplification Hell: Revisiting Network Protocols for DDoS Abuse". In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2014.

[21] Iman Sharafaldin et al. "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy". In: *IEEE 53rd International Carnahan Conference on Security Technology (ICCST)*. 2019, pp. 1–8.

[22] Peter Phaal, Sonia Panchen, and Neil McKee. *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*. RFC 3176. 2001.

[23] Martin Roesch. "Snort – Lightweight Intrusion Detection for Networks". In: *Proceedings of the 13th USENIX Large Installation System Administration Conference (LISA)*. 1999, pp. 229–238.

[24] Open Information Security Foundation (OISF). *Suricata – Open Source IDS / IPS / NSM Engine*. 2024. URL: https://suricata.io/ (visited on 02/01/2025).

[25] Benoit Claise. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954. 2004.

[26] Benoit Claise, Brian Trammell, and Paul Aitken. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. RFC 7011. 2013.

[27] Mohamed Amine Ferrag et al. "Deep Learning for Cyber Security Intrusion Detection: Approaches, Datasets, and Comparative Study". In: *Journal of Information Security and Applications* 50 (2020), p. 102419.

[28] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[29] Toke Høiland-Jørgensen et al. "The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel". In: *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*. 2018, pp. 54–66.

[30] Philippe Biondi. *Scapy: Packet Crafting for Python*. https://scapy.net. 2024.

[31] European Parliament and Council of the European Union. "Regulation (EU) 2016/679 – General Data Protection Regulation (GDPR)". In: *Official Journal of the European Union* (2016).

[32] United Nations. *Transforming our world: the 2030 Agenda for Sustainable Development*. 2015. URL: https://sdgs.un.org/2030agenda (visited on 03/01/2026).

[33] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. *CIC-DDoS2019: A Labeled Dataset for DDoS Attack Detection and Evaluation*. Canadian Institute for Cybersecurity, University of New Brunswick. 2019. URL: https://www.unb.ca/cic/datasets/ddos-2019.html.

Appendix A

Alignment with the Sustainable Development Goals

The United Nations 2030 Agenda for Sustainable Development [32] defines seventeen Sustainable Development Goals (SDGs) as a universal framework for addressing global economic, social, and environmental challenges. Although this thesis is a technical work in network security, its contributions relate to several of these goals insofar as reliable, efficient, and open digital infrastructure is increasingly recognised as a prerequisite for sustainable development. The four SDGs most directly aligned with the thesis are discussed below.

SDG 9: Industry, Innovation and Infrastructure

SDG 9 calls for the development of resilient infrastructure, the promotion of inclusive and sustainable industrialisation, and the fostering of innovation. Target 9.1 specifically emphasises the development of reliable, sustainable, and resilient infrastructure, including regional and transborder infrastructure, to support economic development and human well-being, with a focus on affordable and equitable access for all. Target 9.c further seeks to significantly increase access to information and communications technology and strive to provide universal and affordable internet access.

This thesis contributes to SDG 9 by addressing a critical vulnerability in the digital infrastructure that underpins modern connectivity. DDoS attacks represent one of the most disruptive threats to network availability; a detector that operates at line rate without hardware-level packet loss directly strengthens the resilience of the infrastructure through which internet services are delivered. The OctoSketch probabilistic architecture replaces unbounded per-flow state tables with fixed-memory data

structures, enabling the system to absorb high-cardinality attack traffic without degradation, a property that is particularly relevant for the high-capacity backbone links that interconnect the global internet. By demonstrating that effective DDoS detection can be built on commodity hardware using open-source tools (DPDK, LightGBM, CloudLab), the work also contributes to the accessibility dimension of target 9.c: the design is reproducible and deployable without proprietary appliances or closed-source software stacks.

SDG 16: Peace, Justice and Strong Institutions

SDG 16 seeks to promote peaceful and inclusive societies for sustainable development, provide access to justice for all, and build effective, accountable, and inclusive institutions at all levels. Target 16.10 calls for ensuring public access to information and the protection of fundamental freedoms, while target 16.a supports the strengthening of relevant national institutions to build capacity for preventing violence and combating terrorism and crime.

DDoS attacks have been used extensively as instruments of coercion against government portals, election infrastructure, public health services, and critical utility networks. A capable, low-latency detection system reduces the attack surface available to actors who seek to disrupt public access to institutional services. This thesis contributes to SDG 16 by providing a detection architecture that is simultaneously accurate (98.54% weighted F1 on header-only features), fast (first alert within 100 ms), and privacy-compliant: the Sketch-ADV mode never inspects packet payloads, making it deployable in regulated environments where deep packet inspection would be legally or ethically impermissible. The combination of strong detection capability and privacy-by-design is directly aligned with the institutional accountability and fundamental-freedoms dimensions of target 16.10.

SDG 11: Sustainable Cities and Communities

SDG 11 aims to make cities and human settlements inclusive, safe, resilient, and sustainable. Target 11.b focuses on implementing integrated policies and plans towards inclusion, resource efficiency, and resilience to disasters. Smart city deployments increasingly rely on low-latency internet connectivity for critical urban services, including traffic management, emergency response coordination, public transport monitoring, and utility grid control.

The dependability of these services is directly contingent on the availability of the underlying network infrastructure. A DDoS attack against a smart city's connectivity

layer can delay emergency dispatch, disrupt real-time transit information, or interrupt utility telemetry. The line-rate detection system developed in this thesis, capable of sustaining operation at 16.90 Gbps with zero packet loss, provides the kind of infrastructure-level protection that smart city deployments require. Its bounded memory footprint and low per-packet CPU overhead make it deployable as an always-on inline function on the same commodity hardware that hosts other city-network services, without requiring dedicated appliances.

SDG 17: Partnerships for the Goals

SDG 17 emphasises the importance of strengthening global partnerships for sustainable development, including partnerships for knowledge sharing, technology transfer, and the promotion of open and accessible data. Target 17.6 calls for enhanced international cooperation on and access to science, technology and innovation, while target 17.16 promotes multi-stakeholder partnerships that mobilise and share knowledge and expertise.

This thesis was conducted within a collaborative academic framework that is itself a model for SDG 17 partnerships. The experimental infrastructure relies on CloudLab [16], a publicly funded open research testbed that provides reproducible high-speed networking environments to researchers worldwide without access barriers. The detection system is built entirely on open-source components (DPDK [3], LightGBM [15]) and evaluated on a publicly available dataset (CIC-DDoS2019 [33]), ensuring that the results are reproducible and the methodology is transferable to other research groups regardless of institutional resources. The supervisor's affiliation with the University of Nebraska–Lincoln and the author's affiliation with Universidad Pontificia Comillas ICAI exemplify the kind of international academic partnership that SDG 17 seeks to foster.

Appendix B

Feature Descriptions

This appendix provides a complete description of the features used in each extraction mode. For each feature, the table lists the group it belongs to, how it is obtained, and whether it is an exact counter or a sketch-derived estimate.

B.1 DPI-Sketch Features (75 features)

Feature	Group	How obtained	Type
total_packets	Raw counters	Total packets received in the 50 ms window	Exact
total_bytes	Raw counters	Total bytes received	Exact
udp_packets	Raw counters	Packets with IP protocol = UDP	Exact
tcp_packets	Raw counters	Packets with IP protocol = TCP	Exact
icmp_packets	Raw counters	Packets with IP protocol = ICMP	Exact
syn_packets	Raw counters	TCP packets with SYN flag set	Exact
http_requests	Raw counters	TCP packets to/from port 80 or 443 with payload	DPI
dns_queries	Raw counters	UDP packets to port 53 with payload	DPI
baseline_packets	Raw counters	Packets sourced from 10.10.2.x subnet	Exact
attack_packets	Raw counters	Packets sourced from 10.10.3.x subnet	Exact
udp_tcp_ratio	Basic ratios	udp_packets / tcp_packets	Derived
syn_ratio	Basic ratios	syn_packets / total_packets	Derived

Feature	Group	How obtained	Type
baseline_attack_ratio	Basic ratios	baseline_packets / attack_packets	Derived
bytes_per_packet	Basic ratios	total_bytes / total_packets	Derived
ntp_monlist	Protocol-specific	NTP packets to port 123 with MONLIST opcode	DPI
ntp_responses	Protocol-specific	NTP response packets	DPI
avg_ntp_resp_size	Protocol-specific	Mean NTP response payload size	DPI
dns_any	Protocol-specific	DNS queries with qtype = 255 (ANY)	DPI
dns_txt	Protocol-specific	DNS queries with qtype = TXT	DPI
dns_responses	Protocol-specific	DNS response packets	DPI
avg_dns_resp_size	Protocol-specific	Mean DNS response payload size	DPI
snmp_getbulk	Protocol-specific	SNMP GetBulk requests (port 161)	DPI
snmp_responses	Protocol-specific	SNMP response packets	DPI
avg_snmp_resp_size	Protocol-specific	Mean SNMP response payload size	DPI
ssdp_msearch	Protocol-specific	SSDP M-SEARCH packets (port 1900)	DPI
ssdp_responses	Protocol-specific	SSDP response packets	DPI
portmap_getport	Protocol-specific	RPC GETPORT calls (port 111)	DPI
portmap_dump	Protocol-specific	RPC DUMP calls	DPI
netbios_name	Protocol-specific	NetBIOS name queries (port 137)	DPI
netbios_dgram	Protocol-specific	NetBIOS datagram packets (port 138)	DPI
ldap_bind	Protocol-specific	LDAP bind requests (port 389)	DPI
ldap_search	Protocol-specific	LDAP search requests	DPI
mssql_sqlbatch	Protocol-specific	MSSQL SQLBatch packets (port 1434)	DPI
mssql_rpc	Protocol-specific	MSSQL RPC packets	DPI
tftp_rrq	Protocol-specific	TFTP read requests (port 69)	DPI
tftp_wrq	Protocol-specific	TFTP write requests	DPI
ntp_amp_ratio	Amplification ratios	avg_ntp_resp_size / 48 (request size)	Derived
dns_amp_ratio	Amplification ratios	avg_dns_resp_size / 60 (request size)	Derived
snmp_amp_ratio	Amplification ratios	avg_snmp_resp_size / 150 (request size)	Derived
total_amp_ratio	Amplification ratios	total amp. requests / total responses	Derived
fragmentation_ratio	Amplification ratios	Reserved (set to 0)	N/A

Feature	Group	How obtained	Type
syn_ack_ratio	Amplification ratios	syn_packets / syn_ack_packets	Derived
syn_only_packets	SYN discrimination	SYN packets with no payload	DPI
http_payload_packets	SYN discrimination	SYN packets followed by HTTP payload	DPI
active_protocols	SYN discrimination	Count of protocols with share $> 0.1\%$	Derived
syn_http_ratio	SYN discrimination	syn_only / http_payload_packets	Derived
syn_only_ratio	Norm. fractions	syn_only_packets / total_packets	Derived
http_payload_ratio	Norm. fractions	http_payload_packets / total_packets	Derived
dns_query_ratio	Norm. fractions	dns_queries / total_packets	Derived
ntp_monlist_ratio	Norm. fractions	ntp_monlist / total_packets	Derived
snmp_ratio	Norm. fractions	snmp_getbulk / total_packets	Derived
ssdp_ratio	Norm. fractions	ssdp_msearch / total_packets	Derived
icmp_ratio	Norm. fractions	icmp_packets / total_packets	Derived
http_request_ratio	Norm. fractions	http_requests / total_packets	Derived
portmap_ratio	Norm. fractions	portmap_getport / total_packets	Derived
netbios_ratio	Norm. fractions	netbios_name / total_packets	Derived
ldap_ratio	Norm. fractions	ldap_search / total_packets	Derived
mssql_ratio	Norm. fractions	mssql_sqlbatch / total_packets	Derived
tftp_ratio	Norm. fractions	tftp_rrq / total_packets	Derived
max_protocol_ratio	Protocol diversity	Maximum value among all normalised ratios	Derived
protocol_diversity	Protocol diversity	Count of protocols with share $> 0.1\%$	Derived
delta_pps_5w	Temporal (sketch)	pps change: current window − window −5	Sketch
delta_pps_10w	Temporal (sketch)	pps change: current window − window −10	Sketch
pps_variance	Temporal (sketch)	Variance of pps over last 20 windows	Sketch
pps_baseline	Temporal (sketch)	Running average pps over ring buffer	Sketch
ratio_vs_baseline	Temporal (sketch)	Current pps / pps_baseline	Sketch
top_ip_pps_50ms	Temporal (sketch)	Heaviest IP pps at 50 ms scale	Sketch
top_ip_pps_1s	Temporal (sketch)	Heaviest IP pps at 1 s scale	Sketch
top_ip_pps_1min	Temporal (sketch)	Heaviest IP pps at 1 min scale	Sketch
ratio_50ms_1min	Temporal (sketch)	top_ip_pps_50ms / top_ip_pps_1min	Sketch
num_heavy_hitters	Temporal (sketch)	IPs exceeding heavy-hitter threshold	Sketch

Feature	Group	How obtained	Type
ip_concentration	Temporal (sketch)	top IP count / total sketch count	Sketch
new_ips_ratio	Temporal (sketch)	Fraction of IPs not seen in previous window	Sketch
attack_entropy	Temporal (sketch)	Traffic entropy over source IP distribution	Sketch
adaptive_threshold	Temporal (sketch)	3-sigma threshold from ring buffer history	Sketch

B.2 Sketch-ADV Features (64 features)

Feature	Group	How obtained	Type
delta_pps_5w	Temporal (sketch)	pps change: current window − window −5	Sketch
delta_pps_10w	Temporal (sketch)	pps change: current window − window −10	Sketch
pps_variance	Temporal (sketch)	Variance of pps over last 20 windows	Sketch
pps_baseline	Temporal (sketch)	Running average pps over ring buffer	Sketch
ratio_vs_baseline	Temporal (sketch)	Current pps / pps_baseline	Sketch
top_ip_pps_50ms	Temporal (sketch)	Heaviest IP pps at 50 ms scale	Sketch
top_ip_pps_1s	Temporal (sketch)	Heaviest IP pps at 1 s scale	Sketch
top_ip_pps_1min	Temporal (sketch)	Heaviest IP pps at 1 min scale	Sketch
ratio_50ms_1min	Temporal (sketch)	top_ip_pps_50ms / top_ip_pps_1min	Sketch
num_heavy_hitters	Temporal (sketch)	IPs exceeding heavy-hitter threshold	Sketch
ip_concentration	Temporal (sketch)	top IP count / total sketch count	Sketch
new_ips_ratio	Temporal (sketch)	Fraction of IPs not seen in previous window	Sketch
attack_entropy	Temporal (sketch)	Traffic entropy over source IP distribution	Sketch
adaptive_threshold	Temporal (sketch)	3-sigma threshold from ring buffer history	Sketch
pps_dns	Per-protocol	DNS sketch total / window_sec	Sketch
hh_dns	Per-protocol	IPs > heavy-hitter threshold in DNS sketch	Sketch
conc_dns	Per-protocol	top_dns_ip / dns_total	Sketch
ratio_dns	Per-protocol	dns_total / global_total	Sketch
pps_ntp	Per-protocol	NTP sketch total / window_sec	Sketch

Feature	Group	How obtained	Type
hh_ntp	Per-protocol	IPs > heavy-hitter threshold in NTP sketch	Sketch
conc_ntp	Per-protocol	top_ntp_ip / ntp_total	Sketch
ratio_ntp	Per-protocol	ntp_total / global_total	Sketch
pps_snmp	Per-protocol	SNMP sketch total / window_sec	Sketch
hh_snmp	Per-protocol	IPs > heavy-hitter threshold in SNMP sketch	Sketch
conc_snmp	Per-protocol	top_snmp_ip / snmp_total	Sketch
ratio_snmp	Per-protocol	snmp_total / global_total	Sketch
pps_ssdp	Per-protocol	SSDP sketch total / window_sec	Sketch
hh_ssdp	Per-protocol	IPs > heavy-hitter threshold in SSDP sketch	Sketch
conc_ssdp	Per-protocol	top_ssdp_ip / ssdp_total	Sketch
ratio_ssdp	Per-protocol	ssdp_total / global_total	Sketch
pps_portmap	Per-protocol	PortMap sketch total / window_sec	Sketch
hh_portmap	Per-protocol	IPs > heavy-hitter threshold in PortMap sketch	Sketch
conc_portmap	Per-protocol	top_portmap_ip / portmap_total	Sketch
ratio_portmap	Per-protocol	portmap_total / global_total	Sketch
pps_netbios	Per-protocol	NetBIOS sketch total / window_sec	Sketch
hh_netbios	Per-protocol	IPs > heavy-hitter threshold in NetBIOS sketch	Sketch
conc_netbios	Per-protocol	top_netbios_ip / netbios_total	Sketch
ratio_netbios	Per-protocol	netbios_total / global_total	Sketch
pps_ldap	Per-protocol	LDAP sketch total / window_sec	Sketch
hh_ldap	Per-protocol	IPs > heavy-hitter threshold in LDAP sketch	Sketch
conc_ldap	Per-protocol	top_ldap_ip / ldap_total	Sketch
ratio_ldap	Per-protocol	ldap_total / global_total	Sketch
pps_mssql	Per-protocol	MSSQL sketch total / window_sec	Sketch
hh_mssql	Per-protocol	IPs > heavy-hitter threshold in MSSQL sketch	Sketch
conc_mssql	Per-protocol	top_mssql_ip / mssql_total	Sketch
ratio_mssql	Per-protocol	mssql_total / global_total	Sketch
pps_tftp	Per-protocol	TFTP sketch total / window_sec	Sketch
hh_tftp	Per-protocol	IPs > heavy-hitter threshold in TFTP sketch	Sketch
conc_tftp	Per-protocol	top_tftp_ip / tftp_total	Sketch

Feature	Group	How obtained	Type
ratio_tftp	Per-protocol	tftp_total / global_total	Sketch
pps_syn	Per-protocol	SYN sketch total / window_sec	Sketch
hh_syn	Per-protocol	IPs > heavy-hitter threshold in SYN sketch	Sketch
conc_syn	Per-protocol	top_syn_ip / syn_total	Sketch
ratio_syn	Per-protocol	syn_total / global_total	Sketch
pps_http	Per-protocol	HTTP sketch total / window_sec	Sketch
hh_http	Per-protocol	IPs > heavy-hitter threshold in HTTP sketch	Sketch
conc_http	Per-protocol	top_http_ip / http_total	Sketch
ratio_http	Per-protocol	http_total / global_total	Sketch
pps_udp_other	Per-protocol	UDP-other sketch total / window_sec	Sketch
hh_udp_other	Per-protocol	IPs > heavy-hitter threshold in UDP-other sketch	Sketch
conc_udp_other	Per-protocol	top_udp_ip / udp_other_total	Sketch
ratio_udp_other	Per-protocol	udp_other_total / global_total	Sketch
mean_pkt_size	Packet size	total_bytes / total_packets	Exact
pkt_size_variance	Packet size	$\mathbb{E}[x^2] - \mathbb{E}[x]^2$ from per-worker accumulators	Derived