XFAST: Efficient Feature Selection for High-Performance Network Traffic Analysis using eBPF/XDP and Genetic Algorithms

Gustavo Henrique Ellwanger Einsfeldt, Alberto E. Schaeffer-Filho
Federal University of Rio Grande do Sul, Porto Alegre, Brazil
Email: {gustavo.henrique, alberto}@inf.ufrgs.br

***Abstract*—Intrusion Detection Systems (IDS) rely heavily on feature-rich data and dimensionality reduction to identify anomalies in high-throughput networks. However, traditional approaches to feature selection are often computationally expensive user-space processes that limit real-time use. In this paper, we present XFAST, a novel in-kernel feature extraction and selection framework built using eBPF/XDP and Genetic Algorithms (GA). XFAST enables low-latency, real-time analysis of network traffic by efficiently computing and refining flow-level features entirely within the Linux kernel. Our system introduces a lightweight, tail-call–based GA execution model that distributes the evolutionary process across multiple packets, using a hit-based fitness function to optimize feature subsets. Our evaluation shows that XFAST improves the F1-score of an Isolation Forest model from 0.80 to 0.85 while maintaining negligible CPU and memory overhead, demonstrating its scalability and efficiency for deployment in edge and cloud-native environments.**

***Index Terms*—eBPF, XDP, Feature Selection, Genetic Algorithm, Intrusion Detection**

I. Introduction

With the continuous evolution of technology, cyberattacks have become increasingly sophisticated and difficult to detect. As a result, Intrusion Detection Systems (IDS) play a critical role in identifying malicious activities within a network and issuing alerts about potential threats [1]. IDS approaches fall into two categories: signature-based detection, which identifies known attack patterns, and anomaly-based detection, which establishes a model of normal network behavior and detects deviations from that baseline.

Anomaly-based methods are more complex to implement due to the challenge of accurately defining what constitutes normal network behavior. However, because they do not rely on previously known attack signatures, they are more effective at detecting zero-day attacks [2]. To establish a reliable profile of normal network activity, it is essential to extract and select relevant features from network traffic. These features serve as the foundation for machine learning (ML) models to perform accurate classification and anomaly detection. In particular, feature selection is a techniques that can reduce high-dimensional traffic while preserving relevant information [3]. Evolutionary approaches, such as Genetic Algorithms (GAs), are promising for this task, as they are able to explore large search spaces and can identify feature subsets that improve detection performance [4].

However, efficiency is a key requirement for continuous and ideally real-time network monitoring. Therefore, both feature extraction and selection should consume minimal computational resources, ensuring ML-based systems can process network data effectively. In this context, the extended Berkeley Packet Filter (eBPF) offers a compelling alternative, enabling programs to execute safely and efficiently within the Linux kernel [5]. With eBPF, programs can run with minimal impact on system performance.

In this paper, we propose XFAST (XDP Feature Adaptive Selection Tool), a system for high-performance traffic feature extraction and selection using eBPF with XDP (eXpress Data Path), designed to enhance anomaly detection capabilities of ML techniques. XFAST includes configurable parameters such as the number of features, precision, sampling rate, and timeout – all adjustable in real time. After extracting the desired features, a genetic algorithm-based feature selector identifies the smallest significant subset of features, through *selection*, *crossover*, and *mutation*, in order to optimize the model's performance and yield the highest anomaly detection accuracy.

The remainder of this paper is organized as follows. Section II presents the background and foundational concepts relevant to the proposed approach. Section III outlines the design goals, addresses the key challenges, and presents an overview of XFAST. Section IV describes the system architecture, outlining both user space and kernel space components. Section V details the prototype implementation, experimental setup, workload, and discusses the evaluation results. Section VI reviews and analyzes related work. Finally, Section VII summarizes the main findings and presents directions for future research.

II. Background

A. eBPF and XDP

The Extended Berkeley Packet Filter (eBPF) [6] is a Linux kernel technology that enables safe, in-kernel programmability without modifying kernel source code or inserting modules. Introduced in version 3.15, eBPF acts as a Virtual Machine (VM) that runs user-defined bytecode on specific system or network events. Prior to execution, programs undergo strict verification to ensure safety, termination, and memory isolation

978-3-903176-75-1

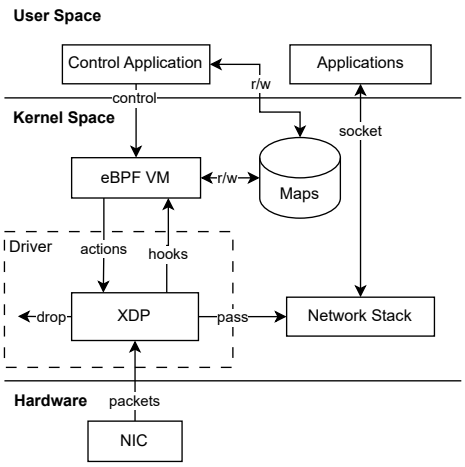

Fig. 1. Architecture of eBPF and XDP.

[7], and then are translated to native instructions by a Just-in-Time (JIT) compiler.

A key abstraction in eBPF is the use of maps, which support shared state between the kernel and user space, enabling efficient data exchange. This model underpins many applications in security, observability, and networking. However, eBPF imposes constraints such as bounded loops, no floating-point support, and limited stack space, which can make complex logic challenging to implement.

An important eBPF-based subsystem is the eXpress Data Path (XDP) [8], introduced in Linux 4.8. It attaches programs at the earliest point in packet processing, within the NIC driver, before memory allocation, as depicted in Figure 1. This approach minimizes overhead, making it well-suited for high-performance use cases such as DDoS mitigation and flow filtering, while being more appropriate for packet header analysis than for payload inspection [9].

In summary, eBPF and XDP provide a robust foundation for building safe, efficient, and flexible in-kernel applications, allowing advanced programmability without compromising performance or security.

B. Feature Selection

Feature selection (FS) is a key data pre-processing step aimed at identifying the most relevant features while eliminating irrelevant, redundant, or noisy variables [10]. By reducing data dimensionality, FS improves computational efficiency, enhances model interpretability, and helps prevent overfitting in machine learning models [11].

FS methods generally fall into three categories: filter, wrapper, and embedded approaches [12]. Filter methods apply statistical criteria independently of the learning model; wrapper methods evaluate subsets using a predictive algorithm, offering higher accuracy at a greater computational cost; and embedded methods integrate feature selection into model training.

The selection task is a combinatorial optimization problem and is NP-hard [13], making exhaustive search impractical in large-scale scenarios. As a result, heuristic and metaheuristic algorithms such as Genetic Algorithms (GA) are frequently used.

In domains like Internet traffic classification [14], FS is especially critical due to the large number of available flow characteristics, not all of which are informative. Selecting a minimal subset improves generalization, and can even outweigh the impact of the classifier choice itself [15].

C. Genetic Algorithm

Genetic Algorithms are adaptive heuristic search methods inspired by the principles of natural evolution [16]. They are particularly effective in solving complex optimization problems, including those involving high-dimensional and combinatorial search spaces. Based on the concept of *"survival of the fittest"*, GAs evolve a population of candidate solutions over successive generations in order to converge toward an optimal or near-optimal solution [17].

This evolutionary process is driven by three main operators: *selection*, *crossover*, and *mutation*, applied repeatedly until a stopping condition is met (Figure 2). Each candidate solution is encoded as a *chromosome*, a binary or integer string where each *gene* represents a problem parameter, such as the inclusion of a feature. A fitness function evaluates the quality of each solution, guiding the evolutionary process by favoring individuals with higher scores.

It begins with a random initial population. In each generation, the following operators are applied:

- **Selection**: Individuals are chosen based on their fitness, often using roulette wheel or tournament selection, favoring higher-scoring candidates for reproduction. To preserve the best solutions, elitism can be applied at this stage, ensuring top-performing individuals are retained across generations [18].
- **Crossover**: Pairs of selected individuals exchange portions of their chromosomes at randomly chosen points, mimicking biological recombination and promoting the combination of beneficial traits.
- **Mutation**: With a low probability, individual genes are altered to introduce additional diversity and explore new regions of the search space, helping the algorithm avoid local optima.

The stopping condition for this iterative process involves reaching a fixed number of generations or meeting predefined convergence criteria. The best solution encountered during evolution is retained as the output.

One of the key strengths of GAs is their population-based search strategy, which enables exploration of multiple regions of the solution space in parallel, reducing the risk of premature convergence. Moreover, GAs do not require gradient information or continuity in the objective function, making them suitable for discrete, non-linear, and multi-objective optimization problems [19].

GAs are particularly suited for feature selection tasks, where each chromosome encodes a subset of features [20]. The fitness function can incorporate classification accuracy or com-

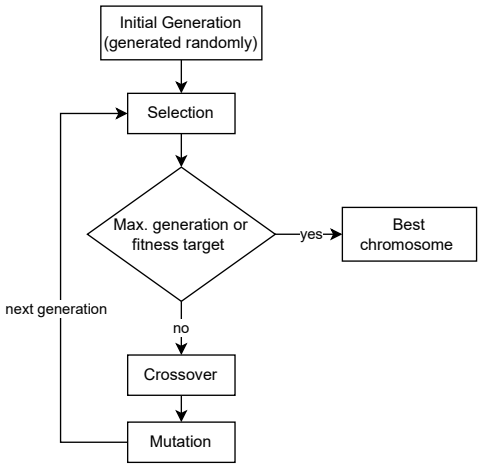

Fig. 2. Genetic Algorithm evolutionary process.

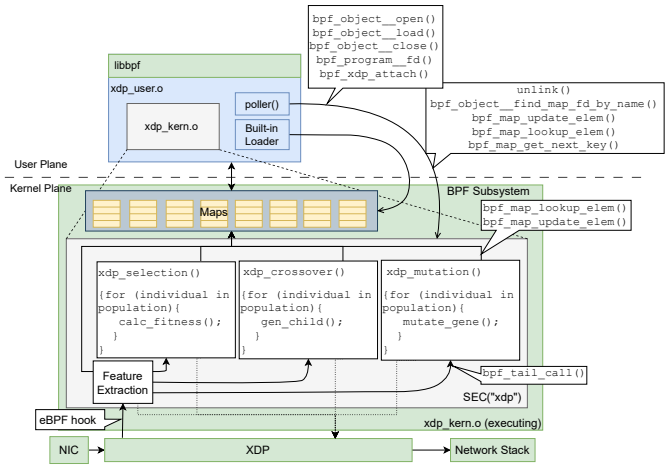

Fig. 3. XFAST approach overview.

putational cost, allowing the algorithm to balance relevance and redundancy.

III. XFAST Design

A. Design Goals and Challenges

Achieving efficient feature extraction and selection for real-time network traffic analysis requires addressing a range of technical and architectural challenges.

Expressiveness of extracted features. One of the primary goals of the system is to enhance feature expressiveness to better characterize traffic flows and improve anomaly detection. However, extending beyond the basic capabilities of the XDP framework introduces significant complexity. Collecting advanced features—such as temporal statistics, inter-arrival times, or multi-packet behavioral patterns—necessitates more intensive computation and frequent interaction with eBPF maps, which can increase system overhead and introduce performance bottlenecks [21].

Processing efficiency and low latency. As the number and complexity of features grow, so does the computational burden on the data plane, requiring a balance between expressiveness and efficiency. The goal is to design a feature extraction mechanism that is sufficiently descriptive for machine learning applications while still maintaining the high throughput and low latency expected from XDP-based programs.

Dealing with limited set of operations. Another challenge is eBPF constraints, such as the absence of floating-point arithmetic, bounded loops, limited instruction sets, and strict verification requirements. These limitations restrict the implementation of many common statistical operations and force the design of optimized, integer-based alternatives. As a result, complex metrics must be reformulated using fixed-point approximations or accumulated over multiple packets, increasing implementation difficulty.

Optimizing GA formulation. The use of GA for feature selection within the eBPF/XDP environment introduces an additional layer of complexity. While GAs are well-suited for exploring large, non-linear, and non-differentiable search spaces, their iterative and population-based nature is computationally expensive—especially in environments with strict per-packet time constraints. A naive implementation of a full GA loop per packet would violate performance expectations and could lead to packet drops or degraded throughput.

To overcome the challenges above, the proposed solution employs a distributed execution strategy based on XDP *tail calls*. The next section will describe the overview of our proposed system.

B. Approach Overview

XFAST (XDP Feature Adaptive Selection Tool) is an efficient system for real-time network traffic feature extraction and selection, designed to support IDSes based on machine learning. XFAST leverages eBPF in conjunction with the XDP framework—both of which offer excellent performance for packet header-based feature extraction and simple operations, thanks to their low-latency and high-throughput characteristics [6].

To enhance the quality of extracted data and eliminate irrelevant or redundant features, XFAST integrates an in-kernel, GA-based feature selection module, as illustrated in Figure 3. To preserve eBPF performance and ensure all computation remains within the kernel, the genetic algorithm is divided into three separate XDP programs: selection, crossover, and mutation. Only one of these runs for each packet, thereby distributing the GA processing across multiple packets. This is coordinated via a persistent state machine stored and updated in eBPF maps, allowing the algorithm to progress incrementally with each packet. In this way, every packet advances the evolutionary cycle, effectively amortizing the computational cost over time.

A custom fitness function was developed to support this design, relying on anomaly-related packet activity (e.g., hit counts) rather than traditional model accuracy metrics or external supervision. This eliminates the need for user-space interaction or auxiliary components. However, this approach introduces specific challenges, such as requiring a minimum

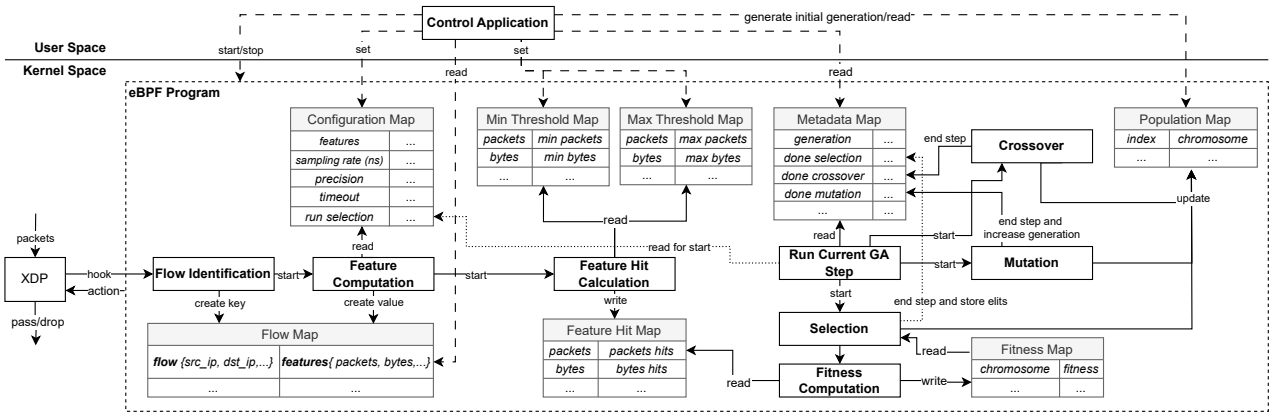

Fig. 4. XFAST system architecture.

packet rate to ensure timely GA convergence. This trade-off was made to maintain all computation within the eBPF/XDP framework, ensuring minimal overhead and real-time performance. The resulting architecture supports adaptive feature selection that responds dynamically to the runtime characteristics of network traffic and IDS alert patterns.

By combining lightweight, kernel-level packet processing with bio-inspired optimization techniques, XFAST offers a promising approach for scalable and deployable ML-based IDS pipelines. The architectural simplicity, native OS integration, and cost-effective execution make it especially suitable for edge-computing and cloud-native scenarios where efficiency, transparency, and minimal resource usage are critical.

IV. System Architecture

The XFAST architecture is divided into two primary components: *user space* and *kernel space* (Figure 4). The user space is responsible for configuring the parameters for feature extraction and selection, initializing the GA population, loading the eBPF program into the kernel via the designated network interface, and consuming the results of extracted and selected features. The kernel space, on the other hand, hosts the core logic of the system, including real-time feature computation and the execution of the GA-based selection process, based on the initial population provided from user space.

A. User Space

The user space component of the system interfaces with the eBPF kernel-space program via BPF maps, utilizing the *map pinning* mechanism. A map pinning binds BPF objects to pseudo-files within the BPF file system, enabling structured and persistent communication between user and kernel space. Through this mechanism, the user space can read from and write to pinned maps in real time, without interrupting the kernel execution.

The system relies on eight pinned BPF maps, each serving a specific purpose in the feature extraction and selection pipeline:

- `Flow Map`: Stores the extracted features for each identified flow on each packet iteration.
- `Configuration Map`: Contains configuration parameters that define how features should be extracted and when the feature selection process should start.
- `Population Map`: Holds the current population of GA individuals, where each chromosome represents a feature subset.
- `Metadata Map`: Maintains the GA's persistent state machine, tracking progress across selection, crossover, mutation, and fitness evaluation stages.
- `Fitness Map`: Records the computed fitness value for each chromosome in the population.
- `Feature Max Threshold Map`: Stores the upper threshold values considered "normal" for each feature.
- `Feature Min Threshold Map`: Stores the corresponding lower threshold values for each feature.
- `Feature Hit Map`: Accumulates, over multiple iterations, the number of times each feature exceeds its predefined threshold range.

The user-space application clears all maps on startup, then initializes the configuration, metadata, threshold, and population maps. After setup, it loads the eBPF program into the kernel and attaches it to the target network interface via the XDP hook.

From there, the kernel program processes packets in real time, updating the maps with feature data and GA results. User space can access these maps at runtime to monitor outputs or adjust configurations dynamically.

More specifically, the *configuration map* enables the user-space application to define a set of operational parameters that govern both feature extraction and selection behavior. These configurations include:

- *Feature mask:* A bitmask indicating which features, among those available in Table I, should be extracted. Each bit corresponds to a specific feature, allowing dynamic selection without modifying the kernel program.
- *Feature extraction interval:* A time interval, in nanoseconds, that determines how frequently features should be recalculated for active flows.
- *Timeout:* The flow timeout value, also specified in nanoseconds, defines the maximum period of inactivity

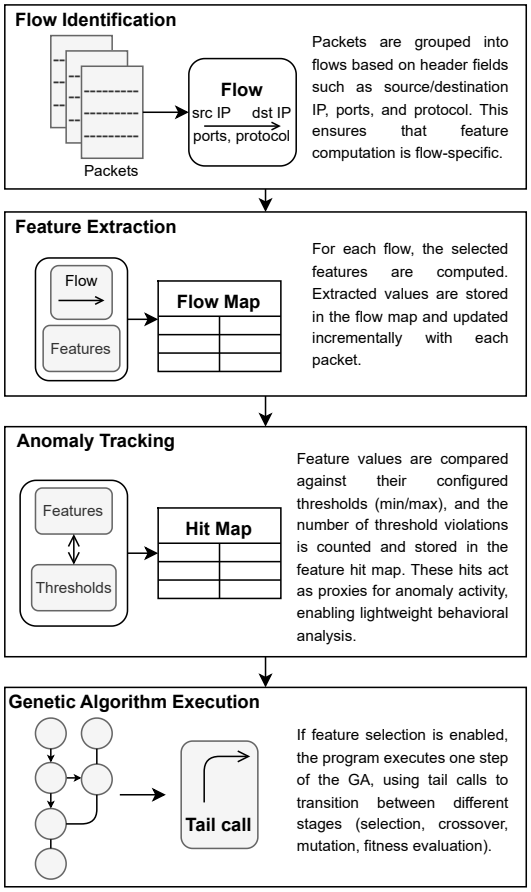

Fig. 5. XFAST core tasks.

allowed for a flow before it is reset from the flow map when a new packet arrives.

- *Precision factor:* A scaling multiplier used to represent decimal values in an integer-only environment. Since eBPF does not support floating-point operations, this parameter enables fixed-point approximations. For example, a precision factor of 10^3 implies that three decimal places are encoded, and values must be divided by the same factor in user space to retrieve their actual representation.
- *Enable feature selection:* A binary flag that indicates whether GA-based feature selection should be activated.

This configuration mechanism allows the kernel-space program to remain generic and reusable, while enabling flexible and context-specific control from user space. Additionally, the use of bitmasking and fixed-point encoding ensures compatibility with the restricted eBPF instruction set, maintaining safety and efficiency in the data path.

B. Kernel Space

The kernel space component is responsible for the core logic of XFAST, implemented entirely using the eBPF infrastructure and attached to the network interface via the XDP hook. Once loaded by the user space application, the eBPF program begins processing incoming packets at the earliest point in the Linux networking stack—directly within the NIC driver, even before memory allocation structures are created.

TABLE I
FEATURES AVAILABLE TO BE EXTRACTED.

Type	Features
Scalar	packets
	bytes
	max packet size
	min packet size
	flow duration (s)
	packets per second
	bytes per second
	inter arrival time (s)
	max packets per second
	min packets per second
	max bytes per second
	min bytes per second
	max inter arrival time
	min inter arrival time
Statistical	average packets per second
	average bytes per second
	average bytes per packet
	average inter arrival time
	variance packets per second
	variance bytes per second
	variance inter arrival time

Each incoming packet triggers the execution of the eBPF program, which carries out the core tasks illustrated in Figure 5. These tasks encompass flow identification, feature extraction, and anomaly tracking, which are executed whenever at least one feature is enabled in the bitmask. The feature extraction mechanism in XFAST is based on our previous work [22]. The execution of the GA is conditionally triggered, requiring both the selection-run flag to be set and the stop parameter in the GA *metadata map* to remain unset.

Feature selection via GA is implemented entirely within the eBPF/XDP kernel context. To address the inherent limitations of the eBPF environment, it relies on *tail call chaining* and persistent state tracking. Each GA stage is encapsulated in a separate eBPF program stored within a program array map, allowing execution to progress across stages through tail calls. Continuity is maintained by a state machine in the *metadata map*, which records the current stage and generation number.

The configuration of the GA is governed by a set of predefined constants that control its behavior and convergence process. These parameters are shown in Table II and must be chosen by the user to balance convergence speed, search space coverage, and processing overhead per packet.

The operation of each GA stage is described in the following items.

1) ***Selection stage****:* In this stage, the fitness of each individual (*chromosome*) in the current population is evaluated based on its ability to detect anomalous activity, determined by features that exceed pre-configured maximum or minimum thresholds. Each chromosome is represented as a bitmask encoding a subset of features. The fitness function is defined as:

$$\text{Fitness}(C) = \left(\sum_{i=0}^{n-1} \left(\frac{\text{mask}_i(C) \times \text{hits}_i}{\sum_{i=0}^{n-1} \text{hits}_i} \right) + \frac{W}{A(C)+1}\right) \times P \quad (1)$$

TABLE II
GENETIC ALGORITHM PARAMETERS

Parameter	Description
Population Size	The number of individuals (chromosomes) maintained in the population. Each individual encodes a candidate subset of features.
Number of Features	Represents the total number of extractable features, determining the length of each chromosome. Each gene is stored as a single bit, packed into a byte array.
Crossover Probability	Controls the likelihood (in percentage) that a non-elite individual will undergo the crossover process during each generation.
Mutation Probability	Specifies the probability (in percentage) of applying a random gene flip mutation to a non-elite individual after crossover.
Fitness Target	Defines the goal fitness value that, if reached by any individual, triggers early termination of the evolutionary process.
Maximum Generations	Limits the total number of generations the GA will execute to avoid indefinite execution.
Feature Weight	A constant used to apply a penalty function that discourages individuals from selecting too many features.

where *C* represents a chromosome, *n* is the total number of features, and $mask_i(C)$, which takes the value 0 or 1, indicates whether feature *i* is selected in chromosome *C*. $hits_i$ denotes the number of times feature *i* has exceeded its configured threshold (either minimum or maximum), as recorded in the *feature hit map*. W is a constant weight applied to features to reward compact chromosomes by penalizing those with too many selected features. *A(C)* represents the total number of active (selected) features in chromosome *C*, and *P* is the precision parameter used to retain decimal information.

This fitness equation was designed to balance two objectives: maximizing anomaly coverage and minimizing the number of selected features. The first component represents the fraction of anomaly hits captured by chromosome *C*. Since it is normalized by the total number of hits, this term is bounded between 0 and 1, serving as a normalized anomaly-coverage score. The second component is the parsimony term, which penalizes individuals with too many active features. The balance between these two terms is explicitly controlled by the constant *W*, ensuring that parsimony does not dominate unless intentionally configured to do so. The precision factor *P* is applied solely for fixed-point representation inside eBPF and does not affect the relative scaling of the terms. Consequently, the higher the number of anomalies captured by a chromosome's selected features, the higher its fitness. One generation of individuals is evaluated per packet.

Once all fitness values are computed, the system applies *elitism*, a strategy well-suited to eBPF's deterministic and resource-constrained model. The two individuals with the highest fitness scores are identified as elite chromosomes and selected as parents for the next generation. Their indexes are stored in the metadata map for use during crossover, and a control flag (`done_selection`) signals that the selection stage is complete, allowing the next packet to trigger the crossover stage.

*2) **Crossover stage**:* The crossover step generates new individuals by recombining the genetic material of two pseudo-randomly selected parents from the population, while preserving the two elite individuals identified during the selection phase. These elites remain unmodified and are excluded from crossover operations to maintain the best-performing solutions.

For the remaining individuals, the crossover operator is applied probabilistically. A pseudo-random condition, determined by system time and the constant *Crossover Probability* (CP), decides whether crossover occurs. When an individual is selected, its chromosome is replaced with a new one produced from two distinct parent chromosomes, ensuring that neither parent coincides with the offspring index.

A single-point crossover mechanism is employed, with the crossover point deterministically derived from a hash of the system time and the individual index. Genes from the first parent are copied up to this point, and genes from the second parent are copied from the crossover point to the end of the chromosome, promoting the inheritance of mixed traits while ensuring reproducibility in the deterministic eBPF/XDP environment. The resulting chromosome is stored directly in the *population map*, and a control flag (`done_crossover`) is set in the *metadata map* to signal readiness for the mutation phase.

*3) **Mutation stage**:* The mutation step introduces random variations into the population to maintain genetic diversity and prevent premature convergence. Mutation is applied probabilistically and only to non-elite individuals, ensuring that the best solutions from the previous generation are preserved.

For each individual, a pseudo-random condition based on system time and the constant *Mutation Probability* (MP) determines whether a mutation is applied. When selected, a single feature (*gene*) within the chromosome is randomly chosen and flipped (toggled from 0 to 1 or vice versa), introducing new traits that enable exploration of regions in the solution space potentially unreachable by crossover alone.

After all potential mutations are processed, the generation counter is incremented, and the control flags in the *metadata map* are reset to prepare for the next GA cycle. This ensures synchronization across the distributed execution model using tail calls, allowing the GA to operate incrementally across subsequent packet arrivals.

Takeaway: Each packet processed by XDP triggers only one step of one GA stage. After processing it, a tail call transitions to the next appropriate program. The current state, generation count, and per-packet offset are stored in the *metadata map*. This design ensures amortized processing cost per packet and scalability to large populations or feature sets. To complete a full GA selection, a *minimum number of packets* must be received (typically $\geq 3 \times$ number of generations).

V. IMPLEMENTATION AND EXPERIMENTAL EVALUATION

This section details the implementation and experimental evaluation of XFAST. First, we describe the system prototype and the experimental setup. Subsequently, we present the evaluation workload and discuss the experimental results.

A. Prototype and Setup

The XFAST prototype was developed using the C programming language and deployed in a virtualized environment running Ubuntu 24.04.2 LTS. All source code is publicly available on GitHub[1].

The user space component was designed to initialize the system, configure the feature extraction and selection parameters, and retrieve the results printed directly to the terminal. To support the GA-based feature selection process and demonstrate its applicability in ML contexts, an Isolation Forest model was trained using Python and the *scikit-learn* library [23]. Isolation Forest was chosen for its efficiency, unsupervised nature, and ability to handle high-dimensional data without assumptions about feature distribution [24]. It was used to establish minimum and maximum threshold values for each feature, which were then used by the fitness function during the GA execution within the eBPF/XDP environment.

An auxiliary tool in C was also implemented to generate the initial population (i.e., the first generation of individuals) for the GA, which was stored in an eBPF map and used by the kernel-space logic.

The experiments were conducted on a VM hosted on an Intel Core i7-11700 (11th Gen, 8 cores) with 12 GB of RAM, running on Oracle VM VirtualBox. This setup validated the system under controlled, near–real-world conditions.

B. Workload

To evaluate XFAST, we relied on the CIC-IDS-2017 dataset [25]. This dataset emulates realistic traffic generated by 25 users over five consecutive days and contains more than 80 flow-level features extracted with CICFlowMeter. Monday's trace is purely benign, whereas Tuesday through Friday include a wide range of attacks (Brute-Force (FTP and SSH), DoS, Heartbleed, Web exploits, Infiltration, Botnet, and DDoS) recorded in both morning and afternoon sessions.

All five days were first merged into a single file. Infinite values, NaNs, and duplicate rows were removed, after which only benign flows were retained to train the Isolation Forest that provides per-feature minimum and maximum thresholds. Using solely benign traffic allows the model to characterize "normal" behavior without bias toward specific attack signatures. The trained model was subsequently applied to the full dataset, including all attack instances, to supply ground-truth threshold violations for GA fitness computation.

For an in-depth assessment of the GA-based feature selector, we used the trace of *Wednesday, 5 July 2017*, which is dominated by high-volume DoS attacks and therefore stresses the anomaly-detection logic.

The dataset provides over 80 flow-level features, while the XFAST prototype currently supports 21 extractable features. Among these, only 12 are comparable with the dataset, consisting of 9 direct equivalents (Total Fwd Packets, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packets/s, Flow Duration, Fwd IAT Mean, Fwd IAT Max, Fwd IAT Min) and 3 derived features (Bytes/s, Bytes per Packet Mean, IAT Variance) constructed from existing fields to approximate the dataset's byte-rate and variance metrics. For evaluation, we trained an Isolation Forest on this 12-feature set and compared it against the reduced subsets produced by the GA, reporting the resulting F_1-*Score*.

[1]https://github.com/geinsfeldt/XFAST.git

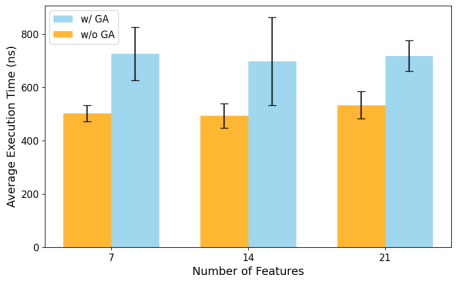

Fig. 6. Impact number of features in the average execution time.

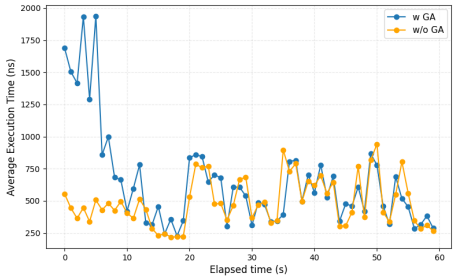

Fig. 7. Average latency over time with 21 features.

To quantify runtime overhead, the Wednesday trace was replayed ten times for 60s per run while varying the number of active features by XFAST (7, 14, and 21). During each run, we measured the average per-packet eBPF execution time (nanoseconds), CPU (%) and memory (%) utilization. During these windows, 680 flows were sent with more than 12k packets and 8000 KBytes in total with 214 packets per second.

C. Experimental Results

The Isolation Forest model trained with the 12-feature set achieved an F1-score of 0.80. After applying the GA-based feature selection with a population of 32 chromosomes during 300 generations, only four features were retained (min forward packet length, flow duration, forward packets per second, and max forward inter-arrival time). Despite the reduced dimensionality, the model achieved an improved F_1-*Score* of 0.85. This result highlights the effectiveness of the GA in eliminating redundant or noisy features while preserving and even enhancing model performance.

As shown in Figure 6, the average execution time of XFAST for different numbers of extracted features (7, 14, and 21) was measured with and without the GA. Increasing the number of extracted features had only a small impact on latency (less than 50 nanoseconds). However, enabling the GA increased the execution time by more than 200 nanoseconds per packet, due to its complexity and map interactions.

Figure 7 shows a deeper look at a single execution with all 21 features enabled during GA execution. The system's

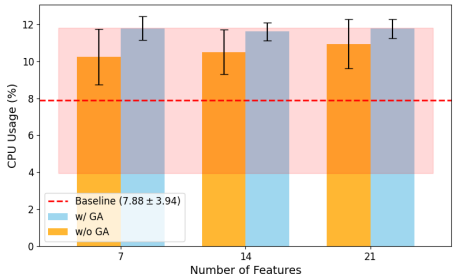

Fig. 8. Impact number of features in the CPU usage.

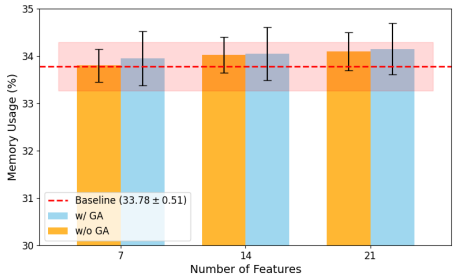

Fig. 9. Impact number of features in the memory usage.

per-packet latency could temporarily spike up to nearly 2 microseconds. After the GA completes its selection, the execution time returns to normal. This confirms that the additional overhead introduced by the GA is temporary and distributed over the processed packets.

In Figure 8, CPU usage remained below 5% of a single core during feature extraction and increased by less than 2% when the GA was active. This slight increase remained consistent regardless of the number of selected features. The low and stable CPU usage demonstrates the system's lightweight nature and its suitability for real-time environments.

Figure 9 demonstrates that the memory usage increased slightly with the number of extracted features due to larger map structures, but the variation remained within the expected system baseline. The overall increase was minimal, even in the worst case. This shows that XFAST maintains a very low memory footprint, which is important for deployment on both edge devices and servers.

These results demonstrate that XFAST can improve detection accuracy while reducing the number of features and maintaining low computational overhead. The system is efficient, scalable, and suitable for real-time network traffic analysis using in-kernel feature selection.

VI. Related Work

Numerous approaches have been proposed for network traffic feature extraction, playing a vital role in applications such as intrusion detection systems (IDS), traffic classification, and performance monitoring. Traditional solutions like NetFlow [26] and IPFIX [27] provide predefined packet-flow-based metrics that support detailed traffic analysis. However, these methods face challenges in high-performance environments, including high resource consumption and limited adaptability due to their rigid architectures.

The emergence of programmable data planes, particularly technologies such as eBPF and P4, has enabled more dynamic and lightweight feature extraction strategies. Recent works have explored these technologies to improve monitoring efficiency and reduce performance overhead. For instance, Magnani et al. [28] proposed an adaptive monitoring system using eBPF, allowing for dynamic metric selection. However, their approach is limited in terms of feature diversity and flexibility. Zhang et al. [29] leveraged eBPF and XDP to extract six flow features and use them for an offloaded neural network IDS, achieving low overhead but facing scalability limitations imposed by eBPF stack size constraints.

Other research efforts have focused on programmable architectures for in-network feature extraction. Silva et al. [30] investigated feature selection techniques for traffic classification in Software-Defined Networks (SDN), while Cugini et al. [31] and Sonchack et al. [32] developed advanced traffic monitoring solutions based on P4. Despite offering high flexibility, these approaches often rely on specialized hardware, limiting their widespread adoption.

Feature Selection techniques have shown significant promise, particularly in scenarios involving large datasets and multi-objective optimization problems [33]. Halim et al. [19] proposed an enhanced GA-based Feature Selection (GbFS) method to improve IDS classifier accuracy. The method was evaluated using three benchmark traffic datasets—CIRA-CIC-DOHBrw-2020, UNSW-NB15, and Bot-IoT—demonstrating superior performance compared to traditional FS techniques, achieving a maximum accuracy of 99.80%.

Similarly, Almaiah et al. [34] conducted a comparative study of FS techniques including GA, Sequential Forward Selection (SFS), and Sequential Backward Selection (SBS), applied to IDS tasks. Their experiments using the NSL-KDD, CIC-IDS 2017, and CIC-IDS 2018 datasets showed that combining GA with classifiers such as SVM and MLP led to higher accuracy and lower false positive rates.

These studies demonstrate the effectiveness of GA in feature selection for intrusion detection, highlighting their ability to improve classifier accuracy while reducing computational complexity. However, most existing feature selection techniques operate outside the data plane, introducing potential latency and limiting real-time applicability. Our work addresses this limitation by integrating a GA-based feature selection mechanism directly into the data plane using eBPF and XDP. This design enables in-kernel, real-time feature selection with minimal overhead, without reliance on external hardware or user-space processing, providing a low-cost and configurable solution suitable for high-performance environments.

VII. Conclusions and Future Work

This paper introduced XFAST, a real-time, in-kernel system for network traffic feature extraction and selection using eBPF/XDP and Genetic Algorithms. By embedding the feature selection process directly in the data plane, XFAST eliminates the latency and overhead associated with traditional user-space approaches. The system employs a hit-based fitness function

and a state-machine–driven GA that executes incrementally across packets, maintaining low per-packet cost while evolving feature subsets. Experimental evaluation on a real-world dataset demonstrates that XFAST enhances anomaly detection accuracy while preserving low CPU and memory utilization under varying configurations. Future work will extend support to more complex feature types, incorporate mechanisms to handle low-volume traffic periods, and provide a richer evaluation of detection quality alongside computational costs. We also plan to broaden experimental validation by leveraging heterogeneous datasets and diverse traffic conditions to strengthen the generalization of the approach.

ACKNOWLEDGMENTS

This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, CNPq (grant #307826/2025-2 and grant #405940/2022-0) and FAPESP (grant #2020/05152-7, grant #2023/00673-7 and grant #2023/00764-2).

REFERENCES

[1] B. L. Dalmazo, J. A. Marques, L. R. Costa, M. S. Bonfim, R. N. Carvalho, A. S. da Silva, S. Fernandes, J. L. Bordim, E. Alchieri, A. Schaeffer-Filho *et al.*, "A systematic review on distributed denial of service attack defense mechanisms in programmable networks," *International Journal of Network Management*, vol. 31, no. 6, p. e2163, 2021.

[2] A. K. Marnerides, A. Schaeffer-Filho, and A. Mauthe, "Traffic anomaly diagnosis in internet backbone networks: A survey," *Computer Networks*, vol. 73, pp. 224–243, 2014.

[3] A. Kalousis, J. Prados, and M. Hilario, "Stability of feature selection algorithms: A study on high-dimensional spaces," *Knowledge and Information Systems*, vol. 12, pp. 95–116, 2007.

[4] S. Ding, C. Su, and J. Yu, "An optimizing bp neural network algorithm based on genetic algorithm," *Artificial Intelligence Review*, vol. 36, pp. 153–162, 2011.

[5] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.

[6] L. K. Organization, "Linux kernel documentation: Bpf," 2014. [Online]. Available: https://www.kernel.org/doc/html/latest/bpf/index.html

[7] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. V. Bernal, "Creating complex network services with ebpf: Experience and lessons learned," in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2018, pp. 1–8.

[8] I. V. Project, "Xdp - express data path," 2018. [Online]. Available: https://www.iovisor.org/technology/xdp

[9] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "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*, 2018, pp. 54–66.

[10] Y. Dhote, S. Agrawal, and A. J. Deen, "A survey on feature selection techniques for internet traffic classification," in *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 2015, pp. 1375–1380.

[11] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[12] D. Carvalho, V. E. Quincozes, S. E. Quincozes, J. F. Kazienko, and C. R. P. dos Santos, "Bg-idps: Detecção e prevenção de intrusões em tempo real em switches ebpf com o filtro de pacotes berkeley e a metaheurística grasp-fs," in *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*. SBC, 2022, pp. 139–152.

[13] T. Naghibi, S. Hoffmann, and B. Pfister, "Convex approximation of the np-hard search problem in feature subset selection," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 3273–3277.

[14] A. Santos da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016, pp. 27–35.

[15] P. Van Der Putten and M. Van Someren, "A bias-variance analysis of a real world learning problem: The coil challenge 2000," *Machine Learning*, vol. 57, pp. 177–195, 2004.

[16] F. U. Khan and S. Bhatia, "A novel approach to genetic algorithm based cryptography," *International Journal of Research in Computer Science*, vol. 2, no. 3, p. 7, 2012.

[17] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[18] D. Beasley, D. R. Bull, and R. R. Martin, "An overview of genetic algorithms: Part 1, fundamentals," *University Computing*, vol. 15, no. 2, pp. 56–69, 1993.

[19] Z. Halim, M. N. Yousaf, M. Waqas, M. Sulaiman, G. Abbas, M. Hussain, I. Ahmad, and M. Hanif, "An effective genetic algorithm-based feature selection method for intrusion detection systems," *Computers & Security*, vol. 110, p. 102448, 2021.

[20] J. Yang and V. Honavar, "Feature subset selection using a genetic algorithm," *IEEE Intelligent Systems and Their Applications*, vol. 13, no. 2, pp. 44–49, 1998.

[21] G. Bertin, "Xdp in practice: integrating xdp into our ddos mitigation pipeline," in *Technical Conference on Linux Networking, Netdev*, vol. 2. The NetDev Society, 2017, pp. 1–5.

[22] G. H. E. Einsfeldt and A. E. Schaeffer-Filho, "NetFeatureXtract: Efficient traffic feature extraction using eBPF," in *International Conference on Advanced Information Networking and Applications*. Springer, 2025, pp. 48–59.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[24] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[25] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani *et al.*, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," *ICISSP*, vol. 1, no. 2018, pp. 108–116, 2018.

[26] Cisco, "Cisco IOS NetFlow Version 9 Flow-Record Format," 2011. [Online]. Available: https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html

[27] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[28] S. Magnani, F. Risso, and D. Siracusa, "A control plane enabling automated and fully adaptive network traffic monitoring with ebpf," *IEEE Access*, vol. 10, pp. 90 778–90 791, 2022.

[29] J. Zhang, P. Chen, Z. He, H. Chen, and X. Li, "Real-time intrusion detection and prevention with neural network in kernel using ebpf," in *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2024, pp. 416–428.

[30] A. S. Da Silva, C. C. Machado, R. V. Bisol, L. Z. Granville, and A. Schaeffer-Filho, "Identification and selection of flow features for accurate traffic classification in sdn," in *2015 IEEE 14th International Symposium on Network Computing and Applications*. IEEE, 2015, pp. 134–141.

[31] F. Cugini, D. Scano, A. Giorgetti, A. Sgambelluri, P. Castoldi, and F. Paolucci, "P4 programmability at the network edge: The braine approach," in *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021, pp. 1–9.

[32] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith, "Turboflow: Information rich flow record generation on commodity switches," in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–16.

[33] A. Lambora, K. Gupta, and K. Chopra, "Genetic algorithm: A literature review," in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. IEEE, 2019, pp. 380–384.

[34] Y. Almaghthawi, I. Ahmad, and F. E. Alsaadi, "Performance analysis of feature subset selection techniques for intrusion detection," *Mathematics*, vol. 10, no. 24, p. 4745, 2022.