FLARE: An FPGA-Based Universal Large Flow Detection Engine

Arish Sateesan[1][0000−0002−8197−0097] ⋆, Jo Vliegen[2][0000−0003−4258−2208], and Nele Mentens[2,3][0000−0001−8753−7895]

[1] iNETS, RWTH Aachen University, Aachen, Germany
[2] COSIC-ES&S, ESAT, KU Leuven, Belgium
[3] LIACS, Leiden University, The Netherlands
arish.sateesan@rwth-aachen.de; {jo.vliegen,nele.mentens}@kuleuven.be

Abstract. Detecting large flows in high-speed networks is a persistent challenge in network security, often hampered by processing speed, memory demands, and the need for versatile handling of a range of attack vectors. The emergence of FPGA-based solutions offers promising prospects for real-time, scalable network security. Yet, precise detection of diverse large flow attacks introduces significant complexity and calls for the coordination of multiple independent detection algorithms. This paper presents FLARE, a large flow detection framework designed to address these challenges by integrating multiple detection algorithms into a unified system. FLARE can monitor network flows in real-time, handling data rates of up to 200 Gbps, and employs a shared architecture that minimizes resource usage while enhancing detection accuracy and coordination. The proof-of-concept implementation on the Alveo U250 data center accelerator shows that FLARE can process an entire packet in every clock cycle, irrespective of the throughput of the employed detection algorithms. Beyond large flow detection, FLARE provides a versatile and scalable platform applicable to a broad spectrum of network security applications.

Keywords: Large flow detection· Heavy-hitter detection · FPGA · DDoS · Network security

1 Introduction

The rapid growth in data rates in recent years has significantly raised the benchmarks for network performance. However, this growth has been paralleled by a rise in cyber threats, particularly volumetric distributed denial of service (DDoS) attacks. Mitigating these attacks, especially volumetric attacks like *large flows*, otherwise called *heavy-hitters*, has become more challenging than ever due to the surge in network speeds and diversity of attack patterns. *Large flows* refer to network flows that consume considerably larger bandwidth than the permitted within a given time frame. These flows can degrade network performance,

⋆ The author carried out this work while at KU Leuven.

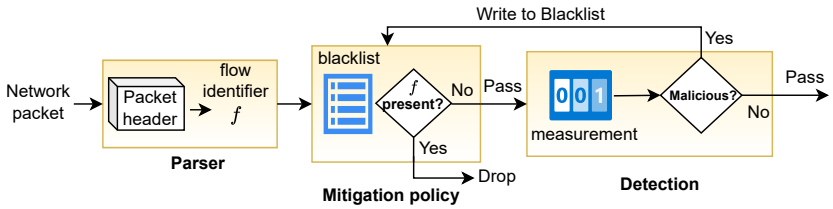

Fig. 1: Generic large flow detection system.

leading to congestion and slowdowns, and may indicate malicious activity like DDoS attacks. *Large flow detection* involves identifying and mitigating these flows, which is a fundamental problem in networking and holds significant importance in many applications, such as volumetric DDoS attack detection and traffic engineering [1–3]. This process typically relies on analyzing network traffic data to identify flows that exceed a set data volume or duration threshold.

Figure 1 shows a real-time flow-based large flow detection system. A network flow is the collection of all network packets that share the same characteristics and are identified by a unique flow identifier (Flow ID). The Flow ID is characterized by the 5-tuple, ⟨source IP address, source port, destination IP address, destination port, protocol ID⟩. In the large flow detection system, a parser extracts the flow ID from the packet header and forwards it to the mitigation and detection blocks. The real-time monitoring results or a predefined set of rules may define the mitigation policy. A commonly used mitigation policy involves both blacklisting [4, 5], maintaining a list of malicious Flow IDs or their fingerprints (f), and dropping any incoming packets that match the blacklist. The detection algorithm determines whether the flow is malicious based on traffic features extracted by the flow measurement unit, where flow measurement refers to the collection and analysis of network flow data.

Previous approaches to large flow detection involve setting a predefined threshold within a specific time frame or measurement epoch. However, this approach struggles to detect a variety of these attack patterns, such as overuse flows [5], which are steady-rate flows slightly exceeding the allocated bandwidth, and burst-flood attacks, which involve periodic transmission of large data bursts within short timeframes [6]. Complete detection capabilities against such diverse patterns often require multiple independent algorithms, increasing computational complexity, resource demands, and coordination challenges. With increasing network speeds, real-time processing demands have outpaced the capabilities of existing detection frameworks. Existing research has yet to offer a universal detection framework capable of coordinating multiple algorithms to manage a broad spectrum of attacks effectively. Universal sketches such as UnivMon [1] and Light-weight Universal Sketch (LUS) [7] take advantage of dedicated measurement modules to collect flow metrics that can be utilized for various other applications. Nevertheless, this approach is orthogonal to the de-

tection of a range of attack patterns of volumetric DDoS attacks, as it focuses only on traffic measurement tasks.

This paper introduces FLARE, an FPGA-based large flow detection engine designed for real-time monitoring and detection of large network flows on high-speed networks, capable of handling data rates up to 200Gbps. Unlike universal sketches like UnivMon and LUS, which aim to generalize detection methods across various applications, FLARE offers a specialized approach by integrating dedicated detection units to identify diverse patterns of network attacks. This positions FLARE as a comprehensive detection system for high-performance data center accelerators, while also broadening its applicability to addressing a wider range of network security challenges beyond large flow detection.

The key contributions of FLARE include:

- A versatile detection framework that effectively integrates and coordinates distinct detection units to identify a wide range of attack patterns, optimized for data center accelerator cards targeting Terabit Ethernet (data rates greater than 100 Gbps).
- A shared architecture that minimizes resource requirements and prevents redundant blacklist entries by facilitating the sharing of blacklists among detection units. Moreover, it uses a probabilistic blacklist, ensuring reduced memory requirement and constant lookup delay irrespective of the blacklist size.
- An independent detection framework and packet forwarding mechanism, which can forward a network packet in every clock cycle, ensuring that the throughput is not affected by algorithm complexity.

2 Challenges in Designing a Large Flow Detection System

Overcoming the challenges of fast and accurate detection of large flows while ensuring efficient hardware deployment involves a multitude of related problems, which will be discussed in the subsequent sections.

2.1 The Need for Dedicated Algorithms

Detecting large flows is a multi-dimensional challenge involving both detection accuracy and range, requiring tailored algorithms for different attack types. The frequent emergence of new and sophisticated attacks, such as carpet bomb attacks [8], underlines this requirement. Large flow detection algorithms typically set a detection threshold based on the average volume over a measurement period. The choice of measurement period and threshold significantly impacts detection accuracy. A long period with a high threshold might overlook potential threats, while a short period with a low threshold risks false positives. Existing algorithms [2, 3, 9] often target large flows that are significantly above (100 to 1000 times) the allocated bandwidth and maintain a steady sending rate. These algorithms, often using simple data structures like sketches [10], tolerate

some measurement errors. While this simplicity of sketches is attributed to lower memory requirements and reduced complexity, it can be exploited by attackers who send traffic just below the high detection threshold, evading detection.

Detecting flows with non-steady rates, such as burst-flood attacks involving periodic transmission of large data bursts within short timeframes [6], introduces additional challenges. Algorithms designed for steady-rate detection may fail to identify bursts, as the average volume over a long period may fall below the threshold. Reducing the measurement period to capture bursts can lead to false positives in steady-rate flows. To accurately detect non-steady-rate flows, the detection algorithm must minimize estimation errors, which can be amplified by sketch-based methods. More sophisticated approaches, such as LOFT [5], can detect flows slightly exceeding the allocated bandwidth, termed as *low-rate overuse* flows. LOFT eliminate the need for distinct detection algorithms to detect low-rate and high-rate overuse flows. ALBUS [6] is another precise detection algorithm designed for detecting burst-flood attacks. These algorithms are less tolerant of estimation errors. However, relying solely on a single detection algorithm is insufficient for comprehensive detection, as LOFT can only detect steady-rate flows, highlighting the need for dedicated algorithms like ALBUS for detecting distinct attack vectors.

2.2 Hardware Deployment and Challenges

FPGAs offer a promising solution for deploying detection algorithms by providing significant parallelism to accelerate computational tasks and meet real-time processing requirements. While graphics processing units (GPU) excel in high-performance computing due to their high throughput design, they aren't optimized for the low-latency processing required by applications like large flow detection. This task demands consistent, real-time performance, and FPGAs can be specifically customized to provide precise, deterministic timing, and highly optimized application-specific hardware designs [11]. Moreover, large flow detection primarily involves operations like data storage and membership queries rather than heavy computations. Transferring data between the CPU and GPU can add extra latency and complexity, whereas FPGAs can be embedded directly into data paths (bump-in-the-wire architectures), thereby minimizing data movement and associated delays [12].

The applications of FPGAs in data centers and networking environment showcases their exceptional potential in high-speed networking applications [13–16]. However, deploying multiple detection units is suboptimal due to the increased resource requirements and complexity, especially in high-speed environments. With blacklisting being the mitigation policy, each detection unit requires a dedicated blacklist, consuming more memory and increasing the detection overhead. Additionally, multiple algorithms identifying the same malicious flow can create redundant blacklist entries, reducing memory efficiency and increasing query latency. The accuracy of large flow detection algorithms solely depends on the accuracy of the measurement module. Limited on-chip memory on FPGAs

can reduce measurement accuracy, as sketch-based units may produce significant overestimations under limited memory. Moving to off-chip memory is not desirable due to considerable access delays, hampering real-time processing [17]. Coordinating algorithms with different throughput and query latencies further complicates hardware deployment, potentially slowing down the system and hindering parallel processing capabilities.

3 Proposed Detection Framework - FLARE

Figure 2 shows the system architecture of FLARE. FLARE is designed to function as a bump-in-the-wire, performing online flow monitoring while positioned between the external network and the protected network to filter traffic. The complete detection engine integrates a parser, a traffic filtering unit, and additional functional and control logic responsible for handling incoming traffic, in conjunction with the detection framework and blacklist module. It buffers incoming packets during processing, forwarding legitimate ones to the network and dropping malicious ones. Further elaboration on the components of FLARE is provided in the subsequent sections.

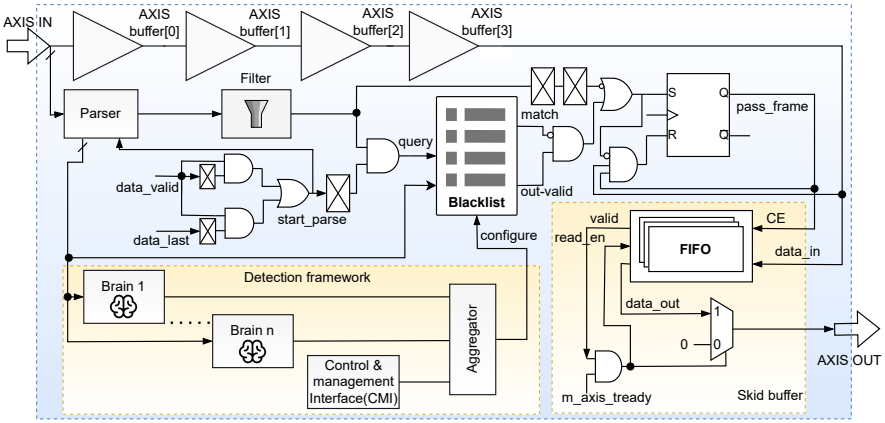

Fig. 2: System architecture of FLARE.

3.1 Parser and Network Filter

The network parser extracts the flow ID and various attributes from incoming flows, such as flow size, EtherType, and protocol. Based on these attributes, the network filter identifies and directs flows meeting the filtering criteria to the blacklist and detection framework. In FLARE, the filtering criteria are based on the EtherType and IP protocol, and it inspects flows with $EtherType{=}IPv4$ and $Protocol{=}TCP/UDP$. In the proof-of-concept implementation of FLARE (Section 4), the protocol used is UDP. Flows that do not match these criteria are

passed through without inspection. The filtering policy is configurable and can be modified during run-time using the control and management interface in the detection framework.

3.2 Detection Framework

The detection framework, the core of FLARE, comprises multiple detection algorithms (referred to as *brains*), a control and management interface (CMI), and an aggregator. Detailed descriptions of these modules are provided in subsequent sections.

Brain: The brain (detection algorithm) determines whether an incoming flow is malicious. Multiple brains can operate concurrently, each tasked with a specific detection role. While brains operate independently, they share a common blacklist, minimizing memory usage. FLARE can also support the sharing of measurement unit, given that the detection algorithms are designed that way, thereby further reducing hardware resources. In such a case, the detection algorithm can function only as a smart decision-maker.

Control and Management Interface (CMI): The CMI is an AXI-Lite module responsible for controlling and validating the detection framework and network filter. CMI also serves the purpose of a brain as it allows users to manually add entries to the blacklist while providing a user interface for testing, framework validation, and blacklist management. The CMI uses a 32-bit control register, along with dedicated data and status AXI slave registers for reading and writing to the blacklist and verifying functionality across modules. The control register also allows dynamic modification of the filtering policy.

Aggregator: The aggregator coordinates interactions among the brains, CMI, and the blacklist. It manages multiple concurrent brains and forwards flow IDs of the detected malicious flows to the blacklist. The aggregator is a smart multiplexer that prevents data loss by ensuring proper sequencing of write operations to the blacklist. The hardware architecture of the aggregator is shown in Figure 3, which can handle three brains and is easily expandable. Data from each brain is queued into the respective data FIFOs (brain B0 is not available, hence set to zero), and the aggregator controls the flow of data to the blacklist based on FIFO status and a predefined priority using a finite state machine (FSM).

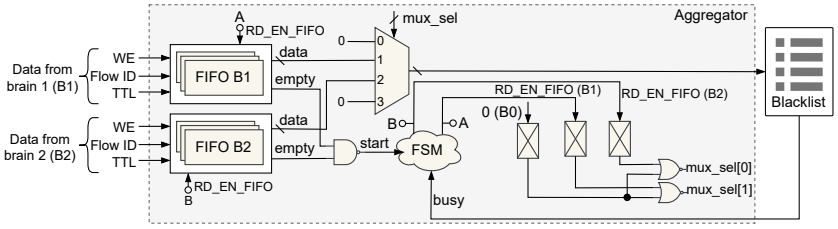

Fig. 3: Aggregator.

3.3 Blacklist Module

The blacklist module consists of a shared blacklist and the associated *time-to-live* (TTL) logic.

TTL Logic: The TTL logic helps to distinguish the flow IDs written to the blacklist by different brains. Each algorithm may have a distinct measurement period or epoch, resulting in varying reset periods. The expiration of each entry on the blacklist depends on the reset period of the corresponding brain. A TTL value, associated with each flow ID, indicates the expiry time of a blacklist entry with respect to a timestamp and is specific for each brain. Figure 4 illustrates the logic circuitry employed in TTL logic. A combination of a prescaler, counter, and an adder generates a value referred to as TTLT (Time-To-Live Timestamp), which is the sum of the current timestamp and the TTL value. This value is stored alongside the flow ID in the blacklist. The prescaler is a clock divider, and it drives a 16-bit counter that produces the timestamp. The size of the TTLT value is taken as 16 bits. The divisor to the prescaler is calculated based on the clock frequency and the size of the TTLT value.

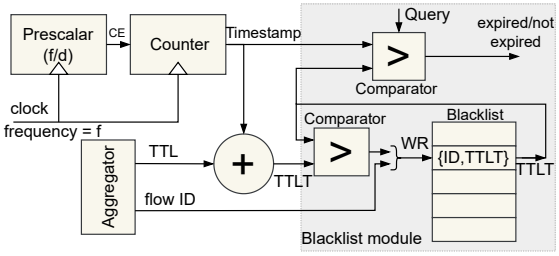

Fig. 4: TTL logic.

Blacklist: The blacklist, implemented using SPArch-based [18] content addressable memory (CAM), consists of a sketch component and a memory unit for TTL values. SPArch is a probabilistic data structure that is used as a counter array, which can be modified as a probabilistic key-value store, a faster alternative to content addressable memories. The architecture of the blacklist is shown in Figure 5. It uses a hash function (Xoodoo-NC [19]) to map flow IDs to the sketch using hash values h_1 to h_d, where d is the number of arrays in the sketch. The size of the hash values is $log_2 m$, where m is the depth of each memory array. The blacklist stores only an 8-bit fingerprint (f) of the flow ID, which is generated using the hash value h_f. Each hash-indexed cell in the sketch stores f and the address A, which points to the TTL memory. The TTL memory stores the associated expiration information (TTLT value). For a detailed description of the update and query operations of the blacklist, we refer to the original work [18].

If a flow ID already exists in the blacklist (previously written by another brain), the existing TTLT value is compared with the new entry, and if the new value is greater, it replaces the old one. To inspect which brain added a specific

flow ID to the blacklist (with the assistance of CMI), an additional identifier tag field of 1 or 2 bits (depending on the number of brains) can be added to the TTL memory. The blacklist is queried for every incoming flow that meets the filtering criteria. If the incoming flow is present in the blacklist but the TTLT value is expired, the flow is allowed to pass and the entry is removed from the blacklist. The address of the removed entry is stored in a FIFO and re-used later. If required, the blacklist can be reset periodically based on the reset periods of the brains.

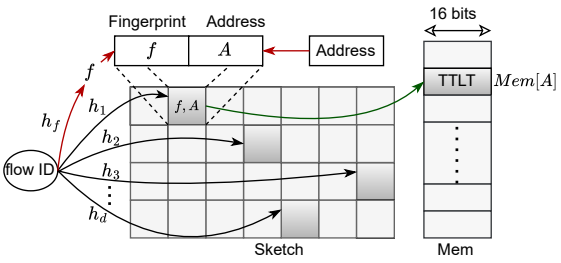

Fig. 5: Architecture of blacklist

3.4 Skid Buffer

The skid buffer, implemented using data FIFOs, buffers outgoing packets until the master AXIS interface is ready. It also handles clock domain crossing. The buffer is controlled by a $pass\_frame$ signal (refer to Figure 2), which ensures that packets are transmitted only after blacklist processing. The $reset$ signal of the SR flip-flop keeps the $pass\_frame$ signal active for one packet, guaranteeing correct transmission behaviour.

4 Implementation of a Proof of Concept

We implemented a proof of concept (PoC) for FLARE to validate its feasibility. The PoC employs the LOFT [5] detection algorithm and CMI as the brains, with LOFT handling large flow detection and CMI enabling manual additions to the blacklist. LOFT, although has a sophisticated design, is chosen due to its superior detection accuracy, which can be as low as 1.5× the allowed bandwidth. While other algorithms, like EARDet [4], also follow the blacklisting model, LOFT was chosen for its efficiency and hardware suitability as it employs sketches for flow measurement. This PoC demonstrates the efficiency of FLARE, and the implementation and hardware evaluation are detailed in this section.

4.1 Hardware Platform

The hardware evaluation was conducted on the Alveo U250 data center accelerator card [20]. Alveo U250 is a robust FPGA-based solution to accelerate data center workloads, supporting data rates up to 200Gbps. Alveo employs two QSFP28 ports as the network interface that can transmit/receive 512 bits per clock cycle. The card has extensive resources: 1,341K LUTs, 2,749K flip-flops, 11,508 DSP slices, 54MB of on-chip memory with 38TB/s bandwidth, and 64GB of off-chip DDR memory with 77GB/s bandwidth. As shown in Figure 6, the Alveo platform is divided into static and dynamic regions. The static region establishes the fundamental framework for the platform, including essential elements such as PCIe connectivity, board management, sensors, clocking, and reset. The dynamic region is subdivided into four super logic regions (SLRs), and are available for user-programmed RTL logic. For a deeper understanding of the Alveo platform, we refer to the Alveo user guide [21].

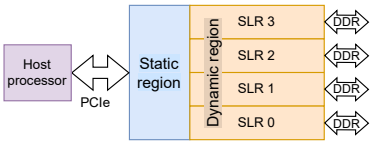

Fig. 6: Alveo U250 floorplan.

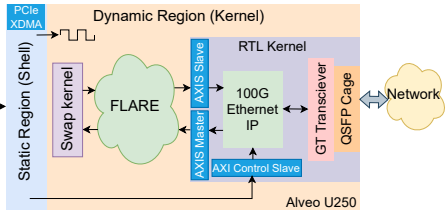

Fig. 7: Evaluation testbed.

4.2 Experimental Setup and Implementation Details

FLARE is integrated into the dynamic region of Alveo U250 as an AXI4-Stream IP, along with the CMAC kernel (100G Ethernet Media Access Control), using Xilinx Vivado 2022.1. The CMAC kernel is adopted from the XUP Vitis network example (VNx) [22], which uses the Ultrascale+ integrated Ethernet subsystem. The CMAC core is built around the CAUI-4 (100 Gigabit Attachment Unit Interface), which uses four lanes operating at 25.78125 Gbps each. This results in an aggregate uni-directional physical layer (PHY) raw data rate of 103.125 Gbps; however, after accounting for the 64b/66b encoding overhead, the effective payload data rate is approximately 100 Gbps. Data from the four lanes is aggregated into 512-bit wide words that are processed at a clock frequency of 322 MHz, the same frequency at which the CAUI-4 interface serializes and deserializes data. By converting high-speed serial data into these wider parallel words, the CMAC core is able to manage bursty traffic more effectively. It achieves this by buffering the data and inserting idle cycles or employing flow control, ensuring that while its instantaneous processing capability might be higher, the average throughput is maintained at the PHY's 100 Gbps limit.

The CMAC kernel offers two 512-bit AXI4-Stream interfaces to connect with FLARE. The setup, shown in Figure 7, connects FLARE to the network via

QSFP and communicates with the host computer through PCI Express. FLARE is interfaced with another AXI4-Stream IP, *Swap kernel*, to verify FLARE's functionality by echoing traffic by swapping source and destination addresses. The system architecture of FLARE is as depicted in Figure 2. The incoming 512-bit AXI Stream data is buffered for four cycles to accommodate the blacklist query data path processing delay, which includes parsing, filtering, and blacklist query. The detection framework, containing the LOFT algorithm and CMI, processes the parsed data, with aggregator facilitating communication between the blacklist and detection components. For the details of LOFT, we refer to the original work [5]. In the implementation, LOFT uses 16,384 24-bit counters (16 bits for flow volume, 8 bits for cardinality), assuming that there are no *jumbo* frames. The size of the flow ID, consisting of the source address and port, is 48 bits. The active flow list and flow table in LOFT are implemented using the Cuckoo hash table, and both have a depth of 16384.

The blacklist, implemented with dual-port block RAM (BRAM) and controlled by two FSMs, handles simultaneous query and configure operations with a query latency of 2 cycles, ensuring processing at line rate. The sketch module of the blacklist has two arrays (d=2, m=1024) and the fingerprint size is 8 bits, adequate to provide near-hundred percent accuracy for storing up to 1024 elements (address size of 10-bits). An 8-bit fingerprint is optimal for SPArch [18], though even a 6-bit fingerprint is sufficient to reach saturation in accuracy. The update operation of the blacklist is not pipelined to conserve resources as well as to improve the operating frequency. As the update operation occurs less frequently, the aggregator can store and queue multiple update requests, making the pipelining of the update operation unnecessary.

4.3 Results

The overall resource consumption of the testbed including the static shell region is summarized in Table 1, detailing the resource usage of various modules. Although the logic resource usage of the testbed is negligible compared to the total available resources, it consumes 32% of the BRAMs. Of this, 20% is allocated to LOFT, while 10% is consumed by the memory subsystem and interconnects. The memory requirement of LOFT, ∼97% of the memory consumption of FLARE, is primarily driven by its measurement module. This underscores the importance of optimizing the algorithm and designing the measurement module to be shared across multiple algorithms so that the memory footprint stays within the constraints of on-chip resources. FLARE requires only <1% of the logic resources and on-chip memory, excluding the detection algorithms.

FLARE has 4 pipeline stages and introduces a total propagation delay of four clock cycles per packet, covering the operations of parsing, filtering, and blacklist lookup to determine whether to forward or drop a packet. The detection framework operates independently and does not interfere with the packet-forwarding operations. The CMAC core can transmit and receive 512 bits per clock cycle at an operating frequency of 322 MHz. Assuming a minimum packet size of 64

bytes, FLARE can process one packet per clock cycle. When metadata is included, an additional clock cycle is required to transmit or receive a 64-byte packet, resulting in a total of two clock cycles for processing. This additional cycle provides enough slack time for FLARE during processing. FLARE supports bidirectional data rates up to 200Gbps, which is the maximum throughput the network interface can support (cf. 4.2). The throughput of packet forwarding remains unaffected by the throughput of detection algorithms, as the detection framework operates independently in parallel. With a network interface that can support higher data rates, FLARE can sustain the required throughput as long as the critical path delay remains sufficient. The detection framework in the PoC implementation can achieve a throughput of 400 million packets per second assuming minimum packet size, consistent with the performance of LOFT. For a detailed discussion of the experimental results of LOFT, we refer to the original work [5].

4.4 Existing Works and Comparison

Exploration of FPGA-based solutions for large flow detection remains limited, with even fewer studies available in integrating DDoS defence algorithms [23,24]. Moreover, most existing solutions are designed for Gbps link speeds, making them unsuitable for Terabit Ethernet environments. Scalability challenges in integrating multiple detection units on hardware further contribute to the underexploration of this research area. The work proposed in [24] incorporates multiple detection techniques for DDoS defense, employing an approach analogous to FLARE. However, their approach targets Gigabit networks and employs filtering techniques rather than independent detection algorithms, making it less suitable for algorithms with variable measurement epochs. Similarly, commercial solutions like Intel's Algo-shield [25] act as an accelerated filtering mechanism that integrates only established defense methods, limiting their adaptability.

Approaches using generic sketches, such as Jaquen [26], Poseidon [27], UnivMon [1], LUS [7], and generalized families of sketches [28], are limited to detecting high-bandwidth flows due to the high overestimation of sketches. These approaches also suffer from high latency and computational complexity [7], hindering their applicability in high-speed environments. Collaborative DDoS defense frameworks like SENSS [29], DefCOM [30], and CoDef [31] enable resource sharing across organizations and networks but introduce coordination complex-

Table 1: FPGA resource utilization.

Design	LUTs	Registers	CLB	BRAM	DSP
Testbed*	**166,646 (9.6%)**	**270,704 (7.9%)**	**46,207 (21.4%)**	**860 (32.0%)**	**9 (0.1%)**
FLARE	21,006 (1.2%)	11,761 (0.3%)	6,212 (2.9%)	547 (20.3%)	0 (0.0%)
Blacklist	1,031 (0.1%)	107 (0.0%)	178 (0.1%)	2 (0.1%)	0 (0.0%)
LOFT	18,341 (1.1%)	9,937 (0.3%)	5,717 (2.6%)	533 (19.8%)	0 (0.0%)
CMAC	9,126 (0.5%)	32,051 (0.9%)	5,838 (2.7%)	17 (0.6%)	0 (0.0%)
Swap kernel	686 (0.0%)	802 (0.0%)	253 (0.1%)	17 (0.6%)	0 (0.0%)

*Including static shell region.

ity and potential privacy risks. These solutions predominantly support Gbps networks, often neglecting algorithmic optimization crucial for low-overhead detection at Tbps speeds.

Recent advancements in FPGA-based data center accelerators, such as Alveo, showed promising trends toward Terabit Ethernet. Recent works on FPGA-based data center accelerators to accelerate workloads show promise [15, 32, 33], but their applications to network security and large flow detection remain underexplored. Data center accelerator card-based smart NICs [34, 35] and 100Gbps network stacks [22, 36] primarily target data analytics, machine learning, and cryptography [37–40]. Nevertheless, these architectures offer valuable insights and can be adapted for large flow detection, significantly reducing design time.

Advantages of FLARE: Compared to the existing solutions, FLARE offers several advantages and those are listed here:
Reduced detection overhead and real-time monitoring: FLARE reduces the detection overhead by sharing a blacklist that operates independently of the detection algorithms. Additionally, it supports sharing flow measurement modules among algorithms with similar measurement criteria, which can further reduce overhead. Since the algorithms and packet forwarding mechanism operate independently in parallel, their complexity does not affect throughput, enabling real-time processing at Tbps link speeds.
Enhanced coordination and communication: TTLT functionality enables easier coordination of individual detection algorithms with different measurement epochs, improving overall efficiency.
Scalability: FLARE can easily accommodate emerging attack detection algorithms without interfering blacklist lookups or packet forwarding throughput, ensuring scalability.
Privacy-preserving: Unlike collaborative defense frameworks, which share resources and data externally, FLARE confines resource sharing internally, addressing privacy concerns.
Robust fault prevention: Compromised detection units (brain) can be isolated and disconnected from the framework in a plug-and-play manner, maintaining robustness and operational integrity.

5 Conclusions

This paper proposed FLARE, an FPGA-based large flow detection engine targeting Terabit Ethernet capable of integrating multiple detection algorithms into a unified framework while sharing a common blacklist. We tested the functionality of FLARE on an ALveo U250 data center accelerator card using high-speed network interfaces, supporting bidirectional data rates up to 200 Gbps. The implementation validates its real-time flow monitoring capabilities and adaptability for high-speed network security applications. Future work will expand its algorithmic repertoire, focusing on shared measurement modules, and explore broader real-world deployments, solidifying its role in next-generation network security.

Acknowledgments

This work is supported by the ESCALATE project, funded by FWO (G0E0719N) and SNSF (200021L_182005), and by Cybersecurity Research Flanders (VR20192203).

References

1. Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with UnivMon. *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, pages 101–114, 2016.
2. T. Yang et al. Elastic sketch: Adaptive and fast network-wide measurements. *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, pages 561–575, 2018.
3. L. Tang, Q. Huang, and P. P. C. Lee. A fast and compact invertible sketch for network-wide heavy flow detection. *IEEE/ACM Transactions on Networking*, 28(5):2350–2363, 2020.
4. Hao Wu, Hsu-Chun Hsiao, and Yih-Chun Hu. Efficient large flow detection over arbitrary windows: An algorithm exact outside an ambiguity region. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 209–222, 2014.
5. Simon Scherrer, Che-Yu Wu, Yu-Hsi Chiang, Benjamin Rothenberger, Daniele E Asoni, Arish Sateesan, Jo Vliegen, Nele Mentens, Hsu-Chun Hsiao, and Adrian Perrig. Low-rate overuse flow tracer (loft): An efficient and scalable algorithm for detecting overuse flows. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 265–276. IEEE, 2021.
6. Simon Scherrer, Che-Yu Wu, Yu-Hsi Chiang, Benjamin Rothenberger, Daniele E Asoni, Arish Sateesan, Jo Vliegen, Nele Mentens, Hsu-Chun Hsiao, and Adrian Perrig. Albus: a probabilistic monitoring algorithm to counter burst-flood attacks. In *2023 42th International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2023.
7. Qingjun Xiao, Xuyuan Cai, Yifei Qin, Zhiying Tang, Shigang Chen, and Yu Liu. Universal and accurate sketch for estimating heavy hitters and moments in data streams. *IEEE/ACM Transactions on Networking*, 2023.
8. Carpet Bomb DDoS Attacks: On the Rise and Evading Detection. `https://www.corero.com/threat-report-carpet-bomb-intro/`, 2023.
9. T. Yang, L. Liu, Y. Yan, M. Shahzad, Y. Shen, X. Li, B. Cui, and G. Xie. Sf-sketch: A fast, accurate, and memory efficient data structure to store frequencies of data items. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 103–106, 2017.
10. G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.
11. Joost Hoozemans, Johan Peltenburg, Fabian Nonnemacher, Akos Hadnagy, Zaid Al-Ars, and H Peter Hofstee. Fpga acceleration for big data analytics: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 21(2):30–47, 2021.
12. Yao Fu. Adaptable Machine Learning with Alveo Data Center Acceleration Cards. `https://www.xilinx.com/publications/events/machine-learning-live/colorado/AdaptableMachineLearning_with_Alveo.pdf`, 2018.
13. Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, et al. Serving dnns in real time at datacenter scale with project brainwave. *iEEE Micro*, 38(2):8–20, 2018.

14. Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. *ACM SIGARCH Computer Architecture News*, 42(3):13–24, 2014.
15. Jagath Weerasinghe, Francois Abel, Christoph Hagleitner, and Andreas Herkersdorf. Enabling FPGAs in hyperscale data centers. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pages 1078–1086. IEEE, 2015.
16. Jagath Weerasinghe, Raphael Polig, Francois Abel, and Christoph Hagleitner. Network-attached FPGAs for data center applications. In *2016 International Conference on Field-Programmable Technology (FPT)*, pages 36–43. IEEE, 2016.
17. Mohamed Hassan. On the off-chip memory latency of real-time systems: Is ddr dram really the best option? In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 495–505. IEEE, 2018.
18. Arish Sateesan, Jo Vliegen, Simon Scherrer, Hsu-Chun Hsiao, Adrian Perrig, and Nele Mentens. SPArch: A hardware-oriented sketch-based architecture for high-speed network flow measurements. *ACM Transactions on Privacy and Security*, 2024.
19. Arish Sateesan, Jo Vliegen, Joan Daemen, and Nele Mentens. Hardware-oriented optimization of bloom filter algorithms and architectures for ultra-high-speed lookups in network applications. *Microprocessors and Microsystems*, 93:104619, 2022.
20. Xilinx. Alveo U250 Data Center Accelerator Card. `https://www.xilinx.com/products/boards-and-kits/alveo/u250.html`, 2023.
21. Xilinx. Vitis unified software platform documentation: Application acceleration development (ug1393). https://docs.xilinx.com/r/en-US/ug1393-vitis-application-acceleration, 2023.
22. Xilinx. XUP Vitis Network Example (VNx). `https://github.com/Xilinx/xup_vitis_network_example`.
23. Yu Chen and Kai Hwang. Collaborative detection and filtering of shrew ddos attacks using spectral analysis. *Journal of Parallel and Distributed Computing*, 66(9):1137–1151, 2006.
24. Cuong Pham-Quoc, Biet Nguyen, and Tran Ngoc Thinh. Fpga-based multicore architecture for integrating multiple ddos defense mechanisms. *ACM SIGARCH Computer Architecture News*, 44(4):14–19, 2017.
25. Stop DDoS Attacks before They Disrupt the Customer Experience. `https://intel.ly/2N9hexa`, 2020.
26. Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. Jaqen: A {High-Performance}{Switch-Native} approach for detecting and mitigating volumetric {DDoS} attacks with programmable switches. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3829–3846, 2021.
27. Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qianqian Li, Mingwei Xu, and Jianping Wu. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *the 27th Network and Distributed System Security Symposium (NDSS 2020)*, 2020.
28. You Zhou, Youlin Zhang, Chaoyi Ma, Shigang Chen, and Olufemi O Odegbile. Generalized sketch families for network traffic measurement. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–34, 2019.

29. Sivaramakrishnan Ramanathan, Jelena Mirkovic, Minlan Yu, and Ying Zhang. Senss against volumetric ddos attacks. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 266–277, 2018.
30. George Oikonomou, Jelena Mirkovic, Peter Reiher, and Max Robinson. A framework for a collaborative ddos defense. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 33–42. IEEE, 2006.
31. Soo Bum Lee, Min Suk Kang, and Virgil D Gligor. Codef: Collaborative defense against large-scale link-flooding attacks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 417–428, 2013.
32. Babak Falsafi, Bill Dally, Desh Singh, Derek Chiou, J Yi Joshua, and Resit Sendag. FPGAs versus GPUs in data centers. *IEEE Micro*, 37(1):60–72, 2017.
33. Christophe Bobda, Joel Mandebi Mbongue, Paul Chow, Mohammad Ewais, Naif Tarafdar, Juan Camilo Vega, Ken Eguro, Dirk Koch, Suranga Handagala, Miriam Leeser, et al. The future of fpga acceleration in datacenters and the cloud. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 15(3):1–42, 2022.
34. Zeke Wang, Hongjing Huang, Jie Zhang, Fei Wu, and Gustavo Alonso. {FpgaNIC}: An {FPGA-based} versatile 100gb {SmartNIC} for {GPUs}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 967–986, 2022.
35. AMD. AMD OpenNIC project. `https://github.com/Xilinx/open-nic/blob/main/OpenNIC_manual.pdf`. Accessed: 2024.
36. Zhenhao He, Dario Korolija, and Gustavo Alonso. Easynet: 100 gbps network for hls. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, pages 197–203. IEEE, 2021.
37. Monica Chiosa, Thomas B Preußer, and Gustavo Alonso. Skt: A one-pass multi-sketch data analytics accelerator. *Proceedings of the VLDB Endowment*, 14(11):2369–2382, 2021.
38. Wenqi Jiang, Dario Korolija, and Gustavo Alonso. Data processing with FPGAs on modern architectures. In *Companion of the 2023 International Conference on Management of Data*, pages 77–82, 2023.
39. Wenqi Jiang, Zhenhao He, Shuai Zhang, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, et al. Fleetrec: Large-scale recommendation inference on hybrid gpu-fpga clusters. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 3097–3105, 2021.
40. Lucas Bex, Furkan Turan, Michiel Van Beirendonck, and Ingrid Verbauwhede. Mining cryptonight-haven on the varium c1100 blockchain accelerator card. *arXiv preprint arXiv:2212.05033*, 2022.