IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 15 October 2022, accepted 17 November 2022, date of publication 1 December 2022,
date of current version 7 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3226269

TOPICAL REVIEW

Extended Berkeley Packet Filter: An Application Perspective

HUSAIN SHARAF, IMTIAZ AHMAD, AND TASSOS DIMITRIOU, (Senior Member, IEEE)
Department of Computer Engineering, College of Engineering and Petroleum, Kuwait University, Kuwait
Corresponding author: Imtiaz Ahmad (imtiaz.ahmad@ku.edu.kw)

ABSTRACT The extended Berkeley Packet Filter (eBPF) is a lightweight and fast 64-bit RISC-like virtual machine (VM) inside the Linux kernel. eBPF has emerged as the most promising and de facto standard of executing untrusted, user-defined specialized code at run-time inside the kernel with strong performance, portability, flexibility, and safety guarantees. Due to these key benefits and availability of a rich ecosystem of compilers and tools within the Linux kernel, eBPF has received widespread adoption by both industry and academia for a wide range of application domains. The most important include enhancing performance of monitoring tools and providing a variety of new security mechanisms, data collection tools and data screening applications. In this review, we investigate the landscape of existing eBPF use-cases and trends with aim to provide a clear roadmap for researchers and developers. We first introduce the necessary background knowledge for eBPF before delving into its applications. Although, the potential use-cases of eBPF are vast, we restrict our focus on four key application domains related to networking, security, storage, and sandboxing. Then for each application domain, we analyze and summarize solution techniques along with their working principles in an effort to provide an insightful discussion that will enable researchers and practitioners to easily adopt eBPF into their designs. Finally, we delineate several exciting research avenues to fully exploit the revolutionary eBPF technology.

INDEX TERMS BPF, eBPF, XDP, Linux kernel, security, network, sandboxing, storage, containers.

I. INTRODUCTION

In recent years, the cloud computing paradigm has grown both in terms of scale and complexity to offer on-demand computing services (software, platforms, hardware) to users over the Internet [1]. Virtualization is one of the key backbone technologies of cloud computing which allows sharing of resources (CPUs, memory and network) in service delivery [2]. In this regard, containers are the most prevalent lightweight virtualization technology in providing cloud services [3]. Additionally, network virtualization has played a vital role in reducing operational costs for service providers by effectively utilizing network resources [4]. Hence, contemporary applications and services increasingly leverage geographically distributed virtualized infrastructures and heterogeneous networking technologies to deliver services that are quite diverse in network resource requirements and performance characteristics with adequate Quality of Service/Quality of Experience (QoS/QoE) to the users [5].

The emergence of micro-services and containerized applications, due to their agility and scalability, enabled efficient hosting and integration of multiple services on a cloud platform [6], [7]. However, the scale, heterogeneity, diversity and dynamicity of micro-services make it a challenging and complex task for service providers to accurately monitor, manage and troubleshoot [8]. Similarly, network function virtualization (NFV) introduces flexibility in managing networks by guiding and allowing rapid deployment of network services such as firewalls, traffic load balancers, and more [4]. Monitoring the performance characteristics of virtual network functions (VNF) along with the instances of NFV to virtualize the hardware makes it critical to guarantee service availability with desired performance and resiliency in case of network failures [9]. Network services are often implemented as service function chains (SFCs) by instantiating multiple containers as micro-services. These SFCs generally are adaptive and dynamic due to the random nature of service

The associate editor coordinating the review of this manuscript and approving it for publication was Jolanta Mizera-Pietraszko.

requirements. Security of cloud resources is also of paramount importance, requiring massive deep traffic observability and analysis for detection of malware, denial of service (DOS) attacks, anomalies, and proper response [10].

However, the traditional monitoring and inspection tools are no longer able to meet today's ever-evolving new environments needs either due to the lack of mechanisms for dynamic and deep fine-grained traffic inspection at arbitrary locations in the cloud platform or due to the severe overhead associated with them [11]. Therefore, these cloud technologies demand a new generation of scalable monitoring, performance measurement and high-fidelity auditing tools that give cloud service providers programmatic visibility over applications, computing and networking infrastructures and devices without impeding system performance [12]. Furthermore, these tools should be easy to develop and deploy in order to satisfy the service level requirements imposed by the growing number of applications such as video conferencing, autonomous driving, cloud gaming, and the emerging world of meta-verse which is regarded as the next-generation Internet paradigm [13].

A promising solution that appeared in recent years has emerged from a filter injected directly into the Linux kernel code known as Berkeley Packet Filter (BPF), originally described by McCanne and Jacobson [14]. The improved version of BPF, known as extended BPF (eBPF), has wide range capabilities for handling generic processing events within the kernel [15]. eBPF is considered to be a general purpose lightweight in-kernel virtual machine (VM) that offers a combination of flexibility, safety and performance, providing a programmable interface for developing and running verifiable custom code inside the Linux kernel at run-time. Thus, applications can gauge the kernel by eBPF programs without any changes in kernel code or affecting other applications with low overhead at run-time. Additionally, Linux containers have become the preferred unit of application management in the cloud, which implies that these two important technologies can work seamlessly under the same environment. Therefore, eBPF represents an ideal monitoring and enforcing mechanism for cloud native services that can be used to collect real-time critical performance metrics related to security policies, resource and network management.

A. SCOPE AND METHODOLOGY

Since its inception in 2014, eBPF has attracted massive adoption both by industry and academia for a wide range of applications such as packet processing for security and networking applications [16], [17], reducing memory and storage access latency [18], [19], and enhancing system performance [20]. However, despite the wide deployment of eBPF, there is no study that covers the application landscape of this innovative technology in emerging cloud applications. So far, there is only a single survey related to eBPF published in 2020 [21] which provides foundational details about the programming interface with the eXpress Data Path (XDP) for fast packet processing. Therefore, to understand the programming details of eBPF, the reader is referred to Vieira et al. [21]. However, since then, eBPF usage has been widened to numerous emerging applications which include machine learning [22], covert channel detection [23], intrusion detection [24], and more (details in Section III). Hence our work is complementary to [21]; our aim is to study eBPF and its key working principles from an application perspective and make it available to a wider audience. It is our belief that this eBPF pioneering technology will have a promising potential to a diverse range of new emerging applications.

It is thus imperative to study the application domains of eBPF thoroughly, empowering researchers, companies, and developers with a clear roadmap of the eBPF landscape. To conduct this study, we primarily searched using the keywords ''extended Berkeley Packet Filter'' and ''eBPF'' on Google Scholar. For each paper found, we first looked at the abstract and then scanned through the paper contents in order to assign it in one of the four key application domains of eBPF: networking, security, storage, and sandboxing. Networking and Security refer to enhancing existing mechanisms or designing new approaches in these areas, Storage refers to speeding up storage operations and shifting local instructions to remote storages, while Sandboxing is about hosting programs in unified environments and analyzing their performance for optimization reasons. We then further consolidated the outcomes of the first round among the authors in order to acquire the set of primary works to be reviewed. This process resulted in a final set of 46 papers distributed as follows: Networking - 21, Security - 17, Storage - 4, Sandboxing - 4. There were also 22 cross papers that overlapped more than one major category in which case these papers were included in the most important category.

For each application category we then analyzed and summarized the solution techniques along with their working principles in order to provide an insightful discussion about performance gains obtained using eBPF and a comprehensive guidebook for eBPF applications. Finally, based on the reviewed use-cases, we point to some of its limitations and highlight several research challenges in order to make the best use of this emerging technology.

B. ORGANIZATION

The rest of the paper is organized as follows; In Section II, we discuss background material, explaining the original BPF and eBPF, and give a brief explanation of the terms and technologies mentioned throughout the paper. In Section III, we categorize eBPF applications and use-cases. Then in Section IV, we discuss the userspace point of view of the presented studies. We describe trends and future research challenges in Section V. Finally, we conclude and summarize our findings in Section VI.

II. BACKGROUND

In this section, we will briefly explain and describe BPF, eBPF, and any related terms to better grasp the technologies described in the paper. To have a better understanding,

a generic architecture is shown in Figure 1 depicting the main system components: in the bottom, we have the hardware equipment (RAM, Network Interface Card (NIC), CPU, SSD, GPU, etc.). The operating system resides on top of that, holding its core component, the kernel (in our case, the Linux Kernel). In this article, we refer to this box by the name kernel space, which includes all other components along with the kernel processes. Then comes the userspace, where the programs and the execution of user processes take place. Finally, we have the user box on the top, which is responsible for the interaction between the user and the system.

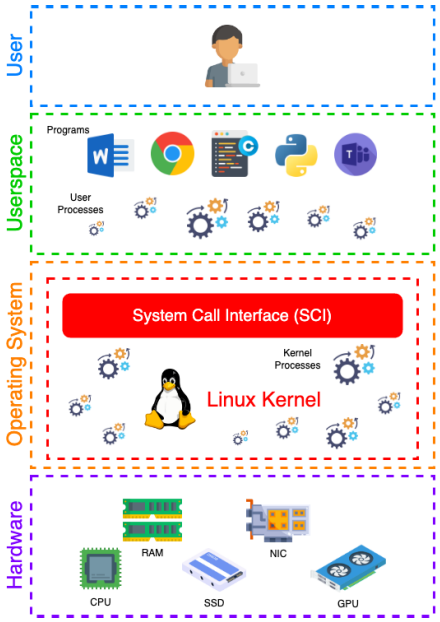


FIGURE 1. Generic system architecture.

Depending on the system we are describing, the existence of the User box and the applications in the userspace might change. For example, a switch in a typical network will handle packets. Thus, there is no interaction with a user as the switch is executing the pre-programmed processes. Hence in this case, the user box (top blue box) and the normal applications will be removed from the picture.

A. BPF

BPF is a kernel architecture designed mainly for packet inspection by McCanne and Jacobson [14]. BPF offered significant performance enhancements at the time, compared with the available packet capture tools. Its objective was to manage and handle the process of copying packets from kernel space to userspace. Kernel space is the operating system's core where it executes and manages the operating system services, while the user space is what a regular user sees and interacts with. The kernel handles the interaction between processes in the userspace, and BPF is a filter placed on the boundaries between the kernel/userspace. The aim was to do in-place filtering rather than moving everything to the userspace. Despite BPF capabilities, it was first introduced to accept/allow or reject/drop packets from entering the network monitoring applications in the userspace. Figure 2 gives a clear overview of how and where BPF is placed in the system.

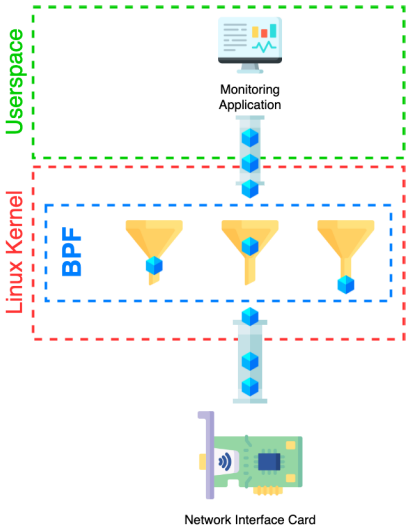


FIGURE 2. Overview of BPF.

The original work [14] demonstrated the performance increase of using BPF over SunOS STREAMS buffering models (NIT) [25] by measuring the overhead produced from moving packets into a buffer before filtering as in SunOS's design or after filtering as in BPF's design. The difference between the two designs is where the filtering takes place. SunOS's design has two buffers, before and after the filter, which means packets are stored in a buffer, then go into the filtering process, and once they pass, they are moved to the second buffer; thus, even unwanted packets are stored and consume a lot of CPU cycles. In contrast, BPF does the filtering process for the stream of packets in-place (once received from network interface), and then stores them in a buffer which means there is no overhead of storing unwanted packets before filtering. The authors tested two basic filtering configurations that will either pass or drop packets. First, they configured the filter to accept all packets (move all to buffer), in which case BPF produced 15 times less overhead compared to NIT. Then they tested the option to filter all packets (drop all), which means that in SunOS's design the packets will be stored in a buffer and then filtered (dropped). At the same time, the BPF design does not pre-store the packets before filtering since they are filtered (dropped) once they pass the network interface as shown in Figure 3. The removal of the pre-storing buffer significantly increased the gain by two orders of magnitude towards BPF favor.

B. eBPF

eBPF [15] is the *extended* version of the classical packet filter BPF. eBPF was defined as a virtual machine that can run sandboxed programs in the kernel. Thus, it is possible to create programs and new filtering applications and host them securely (as a result of the virtual environment and

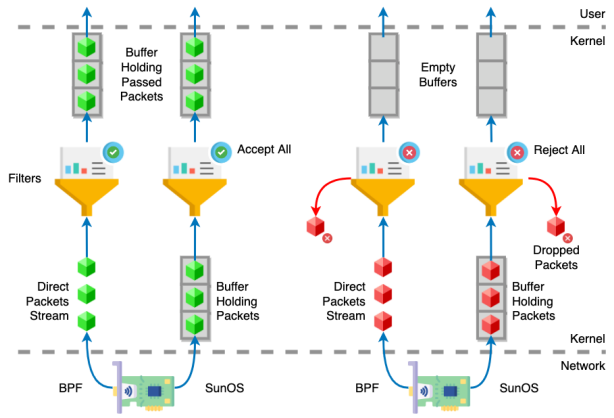


FIGURE 3. BPF and SunOS designs in the two test cases "Accept All" and "Reject All" (i.e. Filter All).

sandboxing) and then append them directly to the kernel code without the need of redesigning the kernel or even rebooting the system. Moreover, it is not limited to kernel applications; developers can execute and run eBPF programs during the system's running time. It is no longer a separate layer or boundary filter like the classic BPF. eBPF is attachable to any of the smaller sub-processes running in the kernel and almost everywhere. Figure 4 illustrates the injection points where eBPF differs from classical BPF. The injection of BPF is limited to the boundaries between userspace and kernel space while on the other hand, eBPF can be placed in any point in the flow. It can be injected between the hardware (storage, NIC) and the kernel space, inside the kernel, on userspace and kernel space boundaries, and even on an application running in the userspace.

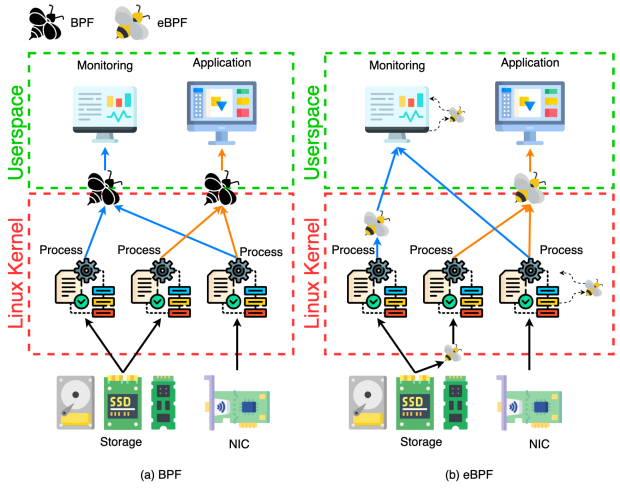


FIGURE 4. Injection of (a) BPF, (b) eBPF.

eXpress Data Path (XDP) [26] is an eBPF-based data path used to intercept packets as soon as they leave the network hardware and before the kernel itself touches the packet data. XDP either drops the packets, or forwards them to userspace (kernel bypass), or passes them to the normal kernel flow (network stack). Injecting eBPF programs means either attaching custom user-defined code or attaching a tracing probe. The tracing probes vary in terms of their name, privileges, mode of operation, type of traced event, and amount of captured data. eBPF is mainly known to use *kprobes* and kernel tracepoints (kernel space tracing for dynamic and static kernel functions) and *uprobes* (userspace tracing for userspace functions) [27].

Figure 5 shows the architectures of BPF and eBPF. BPF consists of two registers: accumulator and index register, an implied program counter, and temporary auxiliary memory. On the other hand, eBPF was expanded from 2 to 11 registers, the size of registers increased from 32 bits to 64 bits, and a stack of size 512 bytes was introduced in the eBPF engine [21]. There are no more limitations to the data size or structure due to a global data map added to the architecture of eBPF.

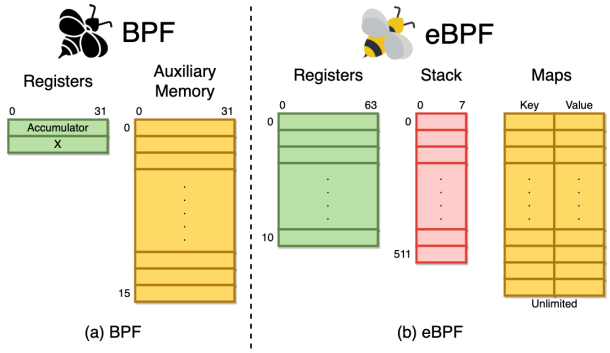


FIGURE 5. (a) BPF architecture, (b) eBPF architecture.

The new version has additional requirements [17] before attaching it to the requested hook while offering new features, ensuring execution safety, efficiency, expandability, and compatibility. Figure 6 summarizes the process of creating an eBPF program and shows the stages required for such a program to be safely injected and correctly executed in the kernel. These stages are described next.

1) VERIFICATION STAGE

A verification stage is required to guarantee the program's safety by validating it according to the following criteria.

a) Privileges: a process holding an eBPF program must have the required privileges to run that program, except when the program does not require any.
b) Run to Completion: a program must run until it finishes, and any loop must have a guaranteed exit condition.
c) Memory out of bounds: programs are prevented from using an infinite number of variables or accessing memory out of bounds.
d) Size: eBPF programs must be limited to a small size to ensure fast and efficient execution.
e) Finite Complexity: filtering mechanisms are based on a Directed Acyclic Graph (DAG), which results in several

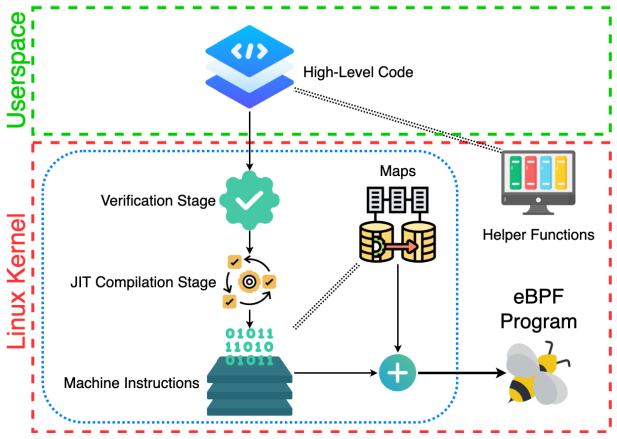


FIGURE 6. Overview of eBPF components and functionality.

execution paths. The verifier must finish the execution within the time limit for all possible paths in the DAG.

f) Crashes: checking that the program will not crash by trying different execution paths.

2) JIT COMPILATION STAGE

Due to the limitation of kernel space resources and the program's complexity constraints, eBPF programs written in Clang or Python must be translated to the correct machine instructions to optimize execution performance. The JIT Compilation stage, following the verification stage, uses the JIT compiler to convert the original version of an eBPF-based program into its optimized machine bytecode version, ensuring the efficiency of the loaded program.

3) eBPF MAPS

In BPF, the limitation of having only a few data structures that can be used in the programs made it extremely difficult for developers to implement more advanced applications. A significant advantage of eBPF is its diversity in data structures. This diversity is based on the concept of eBPF Maps. It is a method of saving the state between invocations of an eBPF program and sending or receiving data between eBPF kernel programs and between kernel and userspace applications. This means storing pairs of keys/values with arbitrary structure. Some examples of the available data structures are hash tables and arrays. A complete list of the data structures is available in the official documentation of the Linux kernel which can be found in [28].

4) HELPER FUNCTIONS

In order to generalize the programmability of eBPF, a stable API has been developed to interact with the kernel, which includes helper functions. The reason for such API is that eBPF programs cannot call normal kernel functions since they will be limited to the kernel version and complicate the programs' compatibility. Some of the most used helper functions are random number generators, time, data, and eBPF map access.

5) TAILING AND FUNCTION CALLS

Despite the fact that eBPF can handle large programs compared to the original BPF, there is still a breaking point and a size limit enforced. Tailing and function calls are employed in eBPF to resolve this issue and enhance the performance of large programs through the clear use of function calls. A large program can be divided into a set of subprograms (function-based), where each function represents a separate component that will be called only if required. The feature can be used in two ways. First, a large program that defies execution time and size constraints can be thought as a chain of ordered staged subprograms (Figure 7(a)). Second, instead of implementing multiple functionalities in a program and executing some of them, it is possible to develop tree-like functionalities such that a function will call another only if needed, thus reducing the amount of executed instructions to only what is important to the program flow (Figure 7(b)). Tailing and function calls enhance the expandability and scalability of eBPF programs to build large applications out of small programs and enhance the overall performance by reducing the amount of executed instructions.

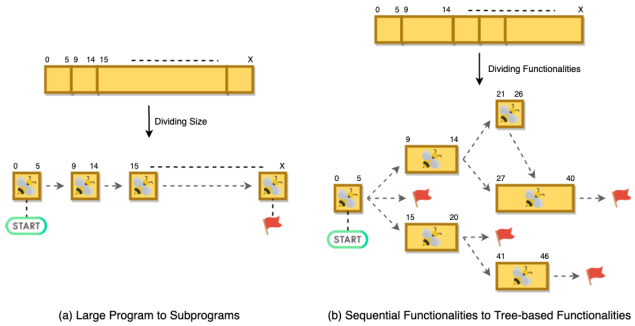


FIGURE 7. Tailing and function calls use-cases. a) Dividing a large program into a chain of staged subprograms. b) Dividing sequential functionalities into a tree-based structure.

III. APPLICATIONS AND USE-CASES

Several papers addressed eBPF and its capabilities, while others discussed a specific use-case or application to compare the performance enhancements with and without the use of eBPF. eBPF is extensively being adopted in both academia and industry. This section summarizes most of the research papers published on eBPF while categorizing the exact use-cases and applications to date. eBPF emerged and became a hot research topic since 2016.

Figure 8 illustrates the use-cases of eBPF in four major areas; Networking, Security, Storage, and Sandboxing. We divided the papers into these major fields according to the objective of each paper. In the papers where objectives overlap (e.g. network security), we classify the paper according to the main objective. When discussing the networking side, eBPF is mainly used for performance improvements, capturing and filtering packets, and controlling the flow of network. In security, the intention is to enhance existing security mechanisms and create novel designs addressing security issues.

Storage related use-cases, focus on increasing memory and storage (HDD, SSD, NVMe) speeds and performance to handle intensive computations, thereby reducing data movement. The final part of the roadmap is sandboxing which focuses on the idea of building better and more suitable, isolated environments for eBPF hosting. In the following, we describe each application domain along with its key use-cases.

A. NETWORKING

Networking hardware is responsible for the communication and interaction among devices in a computer network, whether it is a small home network, a company network or the online services available on the internet. A typical network consists of routing and switching devices, load-balancers, hubs and data servers. All of the mentioned devices process a huge amount of packets. Due to buffer and memory limitations, packets may be dropped when there is heavy traffic in the network, thus pushing companies to get more and better equipment. eBPF technology offers new opportunities for improving network capabilities without the need for extra hardware installations. This section discusses works employing eBPF to enhance packet processing speeds as well as reducing and managing the loads on networking devices.

1) IPTABLES-BASED eBPF

A typical firewall in Linux systems is based on iptables, which is a set of rules handling incoming/outgoing traffic based on IPs and source/destination ports. For example, one of the possible iptables rules that increases security is to allow all outgoing traffic while disallow all incoming traffic. iptables are used to control what IP addresses/ports are visible to the internet, or opening a set of ports for specific hosts (IP addresses) in a local network or subnet. Deepak et al. [29] designed an eBPF-based iptables scheme. They proposed a generic way to translate a given input table rule set (old iptables rules) to an eBPF-based version, by storing the new rules as value-to-action records in the eBPF maps. Therefore, the translation model helped overcome the problem of re-implementing the rules or maintaining a custom code to assist in the migration of old devices' configurations into the new ones.

2) INTER-VM TRAFFIC MONITORING

Monitoring tools and packet analyzers require traffic to be directed to the place in which they are installed. Traditionally each application and tool is installed in a separate VM on separate servers. The traditional way of directing the traffic to the location of these tools was by using a Test Access Port (TAP) device. This is hardware equipment similar to a switch that is installed on top of available servers, replicating the traffic and sending one copy to the desired application server and the other to the monitoring server. Nowadays, as a server can run multiple VMs, the regular application VM and the monitoring application VM, the old TAP is not helpful. Hence a new version of TAP was designed called Virtual TAP (vTAP). vTAP is installed in the server, using virtual switches, replicating the traffic and sending it to multiple VMs. The problem of vTAP is the consumption of host machine resources which can lead to degradation of its performance. Hong et al. [30] designed an eBPF-based vTAP to duplicate the incoming traffic and redirect it to the desired VM as well as the monitoring VM. Testing their design against Open vSwitch (OVS) [31], showed large enhancements in terms of packets per second (PPS), receiver throughput (RX), and CPU usage.

3) LOAD BALANCING

The expandability of data centers and data movement leads to the importance of load balancing. Even a slight enhancement can lead to a large performance increase in the data centers. Chen et al. [32] proved the feasibility of building a load balancer based on machine learning (ML). The data collection stage for building the ML model leverages eBPF capabilities for collecting runtime system data. This work showed the efficiency of using data collected with eBPF to build a new ML model in lieu of the Linux's Completely Fair Scheduler (CFS).

4) IN-KERNEL PROCESSING FRAMEWORK USED IN NETWORK FUNCTIONS VIRTUALIZATION

Network Functions Virtualization (NFV) is the concept of increasing the flexibility of the network equipment regarding packet forwarding and routing by making the network programmable using virtualization technologies [33], [34].

Miano et al. [35] proposed Polycube, a software framework for network functions to run in the kernel. Their objective was to utilize NFV for in-kernel packet processing applications using eBPF. The most known NFV frameworks rely on kernel-bypassing approaches rather than in-kernel processing [36], [37].

5) SRv6 FUNCTIONS FRAMEWORK USED IN NETWORK FUNCTIONS VIRTUALIZATION

With the advancements of Software Defined Networking (SDN) and NFV, it is expected to support programmable interface in network nodes. A key requirement of the rapid innovation and constraints of NFV is to be applied dynamically and to make sure that network operators are flexible for new functions integration. Segment Routing (SRv6) [38], [39] is one of protocols that needs to support network functions. A framework is proposed in [40] utilizing eBPF virtualization to implement network functions in SRv6. The proposed framework provided helper functions that allowed network operators to encode their own network functions as eBPF programs to be automatically executed while processing packets.

6) IN-BAND NETWORK TELEMETRY

In-band network telemetry (INT) is a revolutionized term corresponding to the old known network telemetry. Telemetry refers to the automated process of remotely collecting and processing network information in order to gain visibility and meaningful insights about the state of the network.

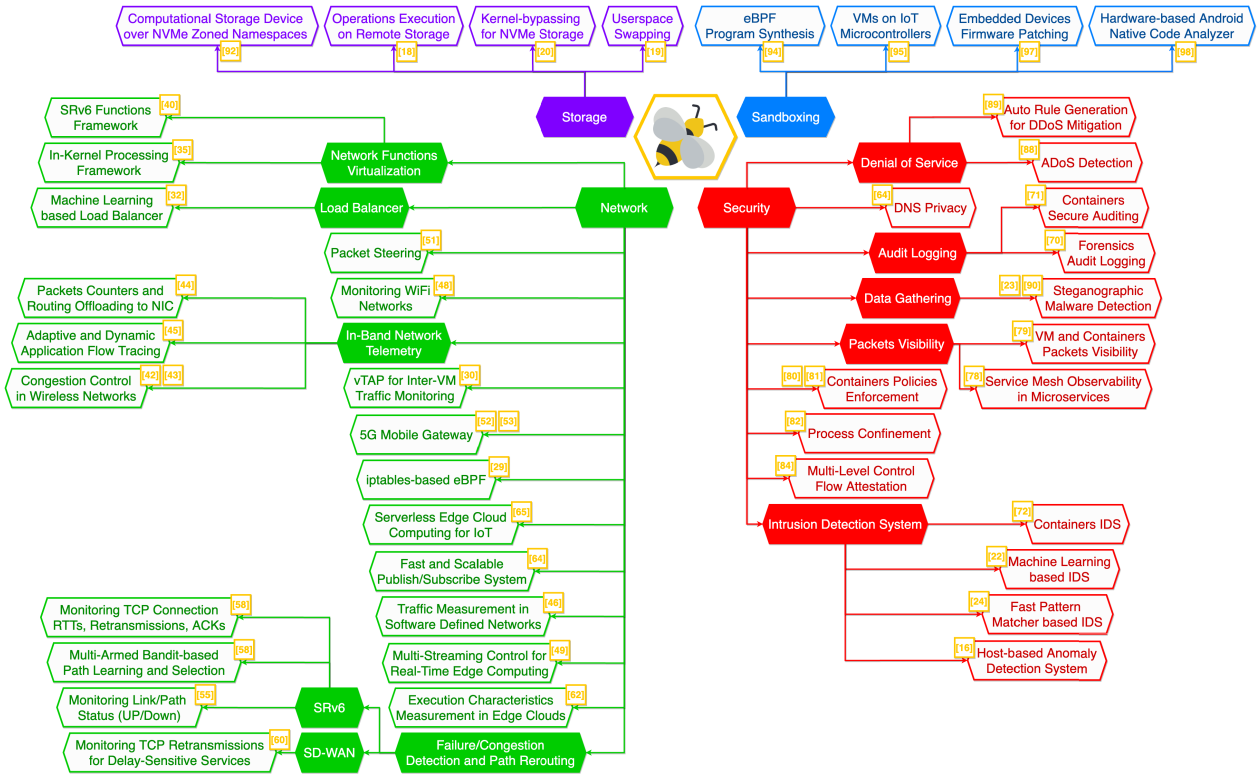


FIGURE 8. eBPF roadmap.

INT enabled network re-orchestration on the level of the packets and on the level of tasks [41].

Bhat et al. [42] discussed the problems of TCP in wireless networks. The current TCP protocol depends on TCP acknowledgements, which may lead to wrong decisions especially when the wireless medium is congested. Bhat et al. designed a new algorithm to handle this problem using a relation between INT data and the window size for the packets. In order to derive the relation, they used the real-time TCP data traced by eBPF in the INT end-to-end option. The results showed throughput improvements of 7 times more than the existing algorithms with less than 20% link loss. Similarly Dong et al. [43] demonstrated eBPF capabilities to detect network congestion at its earlier stages.

SmartNIC enabled the design of telemetry-based network monitoring application for on-going monitoring while reducing the footprint of software network analytics. Abranches et al. [44] implemented a prototype based on eBPF to offload specific pieces of information and computations to the SmartNIC. eBPF maps provide access to SmartNIC statistics. On top of the statistical counters, their scheme uses the HyperLogLog (HLL) algorithm to categorise the paths in order to find unique flows and distinguish them. The authors also designed a routing unit to route the packets to the desired application based on IP-to-application routing tables that are offloaded and stored in the SmartNIC.

Finally, in-band network measurement is an alternative of in-band network telemetry introduced by Sommers and Durairajan [45]. They developed an open-source project code-named ELF for active and dynamic in-band application flow tracing. One of the main problems of current monitoring tools is the rigid flow tracing, where the network operator must define the application routes to be traced, which made them infeasible in case of mid-flow route changes. Similarly, due to load balancing the selected path may affect the throughput of the application. ELF uses eBPF to attach probes accompanying the normal application packets flow, and then dynamically adjust itself. This is done by periodically analyzing a copy of the returned packets to configure the new flow of the application and measure the throughput of the network in order to produce new measurement probes and inject them into the new path.

7) TRAFFIC MEASUREMENT IN SOFTWARE DEFINED NETWORKS

Software Defined Measurement (SDM) was recently introduced to refer to traffic flow monitoring and management in Software Defined Networking (SDN). Zha et al. [46] studied different designs and frameworks to present a handful of comparisons for data centers relying on the deployment of Open vSwitches (OVS) [31]. The authors discussed the architectural design of UMON [47] as shown in Figure (9(a)), which preserves OVS's original architecture while introducing the flexibility of controlling the monitoring tables using APIs and decoupling monitoring (e.g., packet/byte counters) from forwarding in userspace. Moreover, the authors proposed five novel designs, the Flow CAPture scheme (FCAP) and the Sketch based MONitoring (SMON) with the option

of implementing the monitoring feature on-path or off-path, and the final design based on eBPF. The path refers to the kernel path used during receiving a packet until the time the packet leaves the OVS (packet out). FCAP/SMON on-path, illustrated in Figure (9(b)), adds a separate monitoring phase after the kernel flow cache that uses the kernel filter table managed by userspace APIs to check for the wanted packets and then updates a set of tuples that is fetched periodically to the monitoring tables in the userspace to update the counters in the monitoring application. In contrast, the off-path scheme Figure (9(c)) uses a newly added ring buffer cache. Once a packet is received, it is duplicated and forwarded to the ring buffer that is used for the monitoring phase (the same as on-path). The other copy passes through the kernel flow cache and forwarded out immediately, thus achieving a complete separation between monitoring and forwarding. The proposed eBPF design Figure (9(d)) uses two sets of eBPF maps to interface with the monitoring application in the userspace. The eBPF program is attached at Traffic Control (TC) ingress and egress, such that once a packet is received, it checks the eBPF map that is used for storing the monitored hosts (IP of the desired packets) to filter out unwanted packets and updates the second eBPF map used for storing the set of tuples (the stats collection counters). After the filtering phase, the packets proceed with the OVS data path. The main advantage of eBPF over FCAP/SMON is that the monitored hosts can be updated during runtime due to the shared eBPF maps and the low complexity of implementing the eBPF program. Results showed that the eBPF design outperforms the other in terms of implementation complexity. eBPF is entirely independent of OVS architecture and can be loaded and updated at runtime, while the others require recompiling and reinstalling on any update. In terms of latency and memory consumption, the off-path designs win at the cost of high memory consumption due to the ring buffer write operations; without the ring buffer, eBPF has the lead. Overall, each design has its pros and cons and depends on the user to determine which can fit the case more effectively.

8) MONITORING WiFi NETWORKS

Sheth et al. [48] pointed out the importance of gaining deep insights into network operation especially in wireless environments. The authors proposed the first framework called FLIP, leveraging eBPF to monitor WiFi networks in order to solve two issues; accurate wired-to-wireless packet monitoring and energy consumption on access points (AP) stations. eBPF's time-stamping facility enabled accurate measurement of wired-to-wireless packet switching as it goes through multiple layers before being attached to the desired wireless access channel. Besides, APs stations suffer from wasting huge amount of energy due to the awakening process, which is the process of going-to-sleep and awakening-from-sleep. To solve these energy consumption issues, the authors proposed profiling duty-cycle patterns of the stations using eBPF capabilities. Although FLIP was the first of its kind, the results showed an energy consumption reduction

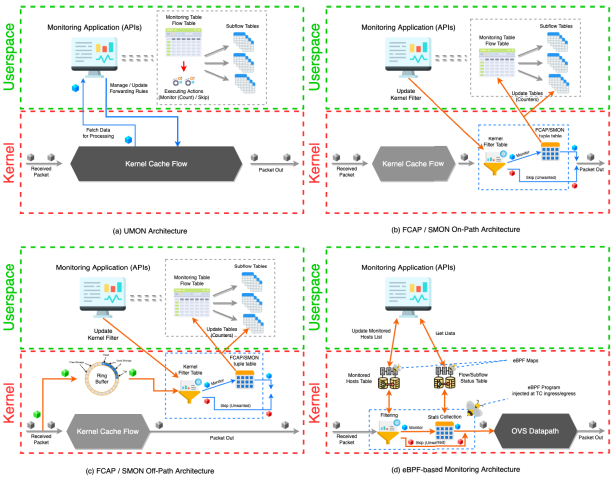


FIGURE 9. (a) OVS-based UMON architecture,(b) On-Path-based FCAP/SMON architecture,(c) Off-Path-based FCAP/SMON architecture,(d) eBPF-based design.

of 6% compared to the standard power monitoring tools. However, the protocol does not require extra hardware installations as regular power monitoring tools do.

9) MULTI-STREAMING CONTROL FOR REAL-TIME EDGE COMPUTING

Data multi-streaming has to be handled by sensors, IoT devices, edge servers, etc. The most important network-related problem is limited capacity due to the congestion caused by replicating data for computation on edge devices. Baidya et al. [49] proposed a computation-aware communication control framework using eBPF for real-time IoT data traffic processed at the network edge. The authors achieved better network and edge server resources utilization, while featuring a dynamic network and packet filtering control.

10) PACKET STEERING

The evolution of network interface cards (NICs) directed researchers to find new approaches for improving Application Level Parallelism (ALP). The difference between CPU and NIC operating speeds caused the system to lag as it cannot keep up with NIC speeds [50]. Even if the CPU speed can handle the traffic (for example 100 Gbps) arriving from the NIC, the memory hierarchy and the small capacity of caches are the bottleneck of the overall process. In [51], a scheme was proposed by combining application level partitioning and packet steering to improve application level parallelism. Figure 10 illustrates the original approach of handling the arrived packets and the new proposed scheme using eBPF XDP program. In Figure 10(a), the packets are stored in a network stack in the kernel and passed to core 1. This core will determine if it is possible to send the packet to other cores in order to distribute the work. The current bottleneck is the speed of core 1 as it needs to process all packets. The proposed approach in Figure 10(b) used eBPF technology to bypass the kernel and distribute the work directly to the available cores. Although this approach requires further

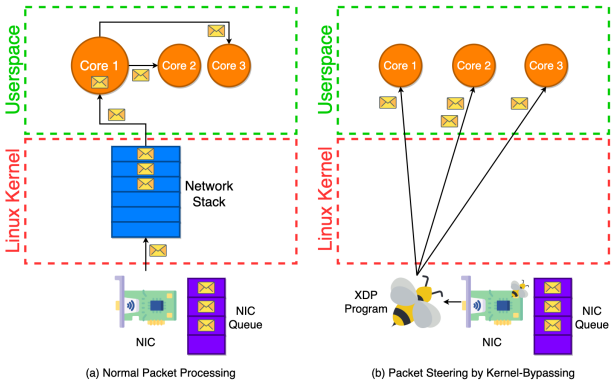


FIGURE 10. (a) The original flow for handling arrived packets, (b) the new approach for packet steering by bypassing the kernel using eBPF XDP program.

evaluation, the authors believe this implementation is a step forward to increase application level parallelism and reducing the overhead of packet processing.

11) 5G MOBILE GATEWAY

Parole et al. in [52] and [53] discussed eBPF integration for 5G mobile gateway in the Mobile Packet Core (MPC). 5G Mobile gateway is responsible for the interconnection of the mobile user devices to Packet Data Networks (e.g. the Internet). The experimental results showed that eBPF can be a good solution for small, distributed centers for edge computing.

12) FAILURE/CONGESTION DETECTION AND PATH REROUTING

Multiple researchers have been studying the area of detecting network path failures and congestion at the time of occurring or even before by analyzing the behavior of the network and predicting the possible upcoming issues. The designs were based on selecting which and what network parameters to capture and use for the action stage, combining multiple parameters, and observing network behavior. The action stage could include alerts sent to the network admin, dynamic rerouting of the flow, data collection for building machine learning prediction models, etc. Based on the experiment environment, we divided the use-cases into Multi-protocol Label Switching (MPLS) and Software-Defined Wide Area Network (SD-WAN).

a: SEGMENT ROUTING WITH IPv6 (SRv6)

SRv6 is a new routing mechanism [39], [54] based on dividing the network into segments and appending routing instructions (labels) on top of the packets rather than maintaining the complete routing paths for each packet on each routing node in the network. The packets will hold the directions (the segments to visit) and traverse the network. On each routing point the packet will use the saved segment directions and the router will point the packet to the next segment routing point. Segment Routing reduces the overhead of the routing nodes individually by only maintaining the information related to the connected nodes, not the entire network.

Xhonneux et al. [55] proposed a new scheme to provide fast-reroute services using eBPF in SRv6. The authors discussed the main three elements (detecting failures, re-route packets on a failure detection and finding backup paths) to provide a fast-reroute service. In their design, the authors classified the network nodes into two types; the master node and the slave node. In Figure 11, we can assume all nodes in segment 1 to be master nodes and the rest are slaves. The master nodes will store the status link (ON/OFF) in the eBPF maps while the slaves will compute the link status when required. The proposed scheme combined with Topology-Independent Loop-Free Alternate (TI-LFA) [56] mechanism leveraging eBPF, leads to a throughput enhancement of 8% compared to classical Bidirectional Forwarding Detection protocol (BFD) [57] that adds an extra application layer protocol and complicates the setup process of the routing paths for the packets (requires a symmetric setup).

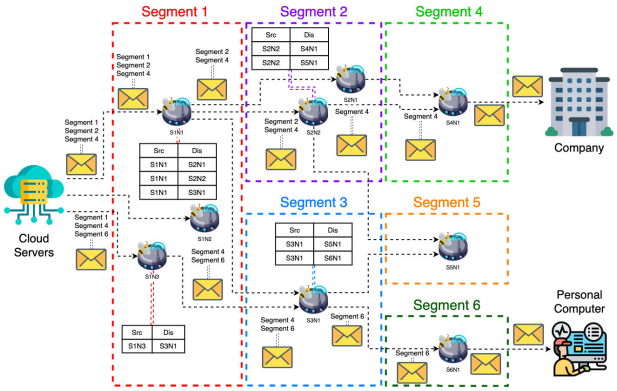


FIGURE 11. Segment routing with eBPF included. Segment 1 routers considered master nodes with eBPF installed. Other segments routers are eBPF-based slave routers.

Jadin et al. [58] proposed an alternative approach to detect failures and reroute paths in SRv6 based on the TCP protocol, called TCP path changer (TPC). The authors divided the tasks into i) recovery from distant link failures; this refers to the detection of route failures from both sender or receiver sides which is then used to select an alternative path, and ii) dynamic selection of lowest-delay paths by monitoring the performance characteristics of the paths. Recovering from path failures requires tracking the path differently from the sender and receiver sides. The sender can be configured to function based on either a counter-based form or a time-based one. In the first type Figure (12(a)), a counter of the number of retransmission timeout messages (RTO) from the sender is maintained and once the counter exceeds a specific number pre-configured by the network designer, a different path is used to re-transmit. In a time-based form Figure (12(b)), the total time of the RTO messages is maintained instead of their actual number. The receiver side, in both cases, works in a counter-based form Figure (12(c)) to count the number of duplicated acknowledgments sent from this side (receiver) to the sender side and use it as a trigger to change the route. When a packet reaches the receiver, it will respond

with ACK to the sender to proceed with the next packet, but if the same packet reaches the receiver, it means that the ACK did not reach out to the sender endpoint because of an issue in the path. Path selection is based on a multi-armed bandit (MAB) [59] approach that balances the exploration of new paths and exploitation of the current paths based on a reward function that captures the implications of the selected paths (collected performance statistics). The available paths are stored in descending order based on their accumulated reward value reflecting the lowest-delay paths. Overall, the proposed approach handles the detection of path failures and then reroutes to the other paths based on the list of lowest-delay paths. TPC runs in an isolated eBPF VM in the kernel space and stores the path headers and the connections on two different eBPF maps. The configurations setup (paths and their requirements) is done by the TPCDaemon, which is a userspace program that works as an interface to the eBPF maps and the eBPF VM hosting the main code.

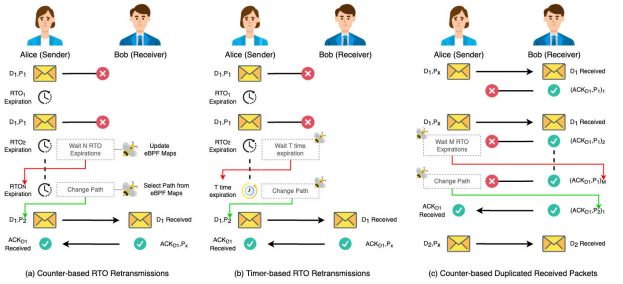


FIGURE 12. **(a) counter-based sender, (b) time-based sender, (c) counter-based receiver.**

b: SOFTWARE-DEFINED WIDE AREA NETWORKING (SD-WAN)

Software Defined Networking (SDN) on a large scale of geographically distanced areas is referred to as Software-Defined Wide Area Network (SW-WAN). Troia et al. [60] designed a new scheme based on eBPF to capture and account for the number of TCP retransmissions in the Open Network Operating System (ONOS) [61]-based WAN controller and deliver them through a buffer to the monitoring applications in the userspace. The proposed design consists of a Kernel Agent (KA) that is the eBPF program, a buffer for storing the captured flow information, and the monitoring application with the pre-defined threshold tables for the number of tolerable TCP retransmissions for each service flow before rerouting the packets flow. The main objective of the design is to increase service resiliency by optimizing and minimizing the downtime caused by network failures (i.e., congestion, off links). The authors claim that the system has a low CPU usage compared to the others, but no actual comparison has been presented. Nevertheless, it can detect real-time segment losses, and no pre-installed software is required as it depends on the eBPF in the Linux operating systems.

13) EXECUTION CHARACTERISTICS MEASUREMENT IN EDGE CLOUDS

Gowtham et al. [62] designed the MessTool framework to profile end-to-end behaviour of distributed applications in cloud developments. The aim of the framework is to help in measuring the performance of cloud infrastructures considering application provider requirements, in a way to provide insights to design specialized cloud environments for designated purposes. MessTool leveraged eBPF capabilities of event tracing and time-stamping facility in the kernel to measure the execution characteristics of an event along the path of execution.

14) FAST AND SCALABLE PUBLISH/SUBSCRIBE SYSTEM

The Publish/Subscribe architecture [63] shown in Figure 13(a) can be used to provide notification services based on user interests. Publishers (the agents responsible for publishing) generate the notifications, and the subscribers (the agents that consume) collect what they are interested in (subscribed for). This scheme is based on intermediate devices called brokers (switch-like middleware). Brokers implement a forwarding-like table for matching notifications and subscribers running on the application layer. An application of this idea can be found in IoT.

In IoT, the sensors (IoT devices) are pre-programmed to collect data. These data are sent to the broker (the middleware). The broker will forward the data to its subscribers' list based on their subscription (their interests). Tatarski et al. [64] presented a novel architecture for Publish/Subscribe scheme that reduces the overhead of running the matching-forwarding process in the application layer (user space) to take place in the kernel level by utilizing eBPF. The authors suggested splitting the broker into conventional edge-broker (CEB) and eBPF-based cloud broker (eBPF-CB) as can be seen in Figure 13(b). CEB pre-processes the notifications (decoupling, grouping, and labeling) as the ordinary broker does and then forwards them to the eBPF-CB. Since the data is already pre-processed, the objective of the cloud broker is limited to forwarding packets only. eBPF's role is to intercept the packets as soon as they arrive, replicate and replace their destinations IP and forward them to the corresponding subscribers. The authors evaluated the notifications forwarded per second for the user space broker and the eBPF-CB. Results show that the latter outperforms the userspace broker by a factor of 20 to 25. Generally speaking, the idea was to divide the original broker task into two subtasks, processing and forwarding. Although the authors only addressed the forwarding task and their approach has significantly increased the forwarding speed and the scalability of the design, they did not evaluate either the complete path latency since they introduced a new processing stage which may add extra latency, or the overall power consumption and its effects on the lifespan of the edge broker and the cloud broker.

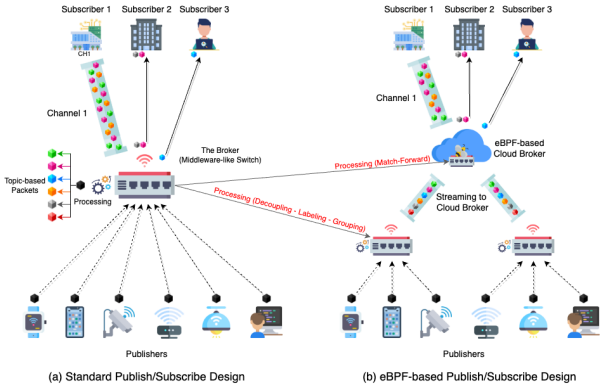


FIGURE 13. (a) The standard publish/subscribe design, (b) the proposed eBPF-based publish/subscribe design.

15) SERVERLESS EDGE CLOUD COMPUTING IN IoT

Serverless Edge Cloud (SEC) is a new cloud paradigm to move computations in cloud architectures closer to the edge-end. Although the main advantage of cloud architectures is its low latency and centralized computation, which can be easily managed and scaled with the available resources, cloud computations suffer from the high cost of maintaining the quality of the offered service. The main goal of SEC is to reduce the cost while maintaining the same service quality. Wang et al. [65] proposed a new SEC-based design for IoT traffic using the Knative framework [66] to solve two main issues i) the cold-start problem of SEC designs and ii) the overhead of Knative queue proxy. The standard protocol used in IoT communications is the Message Queueing Telemetry Transport (MQTT) [67], which is based on the design patterns of publish/subscribe scheme of Figure 13(a). On the other hand, Knative uses HTTP for its communications. Thus, the first thing the authors introduced is an MQTT-to-HTTP adaptor that converts from standard MQTT protocol to HTTP protocol for Knative to be able to handle it. The cold-start problem occurs when a service goes into sleep mode due to an extended period of inactivity which results in in high response latency when a request arrives and the service becomes functional again. The authors designed a prediction component for requests arrivals based on XGBoost [68]. The authors leveraged eBPF to design an event-driven proxy (eProxy) to replace the Knative queue proxy. The eProxy will let the CPU stay in its idle state without wasting any cycle. It is placed on top of the ethernet interface; therefore, on predefined events (e.g., packets arrivals, a chain of packets), the eProxy will be triggered, and the CPU will start processing. Results showed that replacing the eProxy with the queue proxy can save up to 37% of the CPU overhead. However, more experiments are needed to study the latency introduced by the new intermediate components (MQTT-to-HTTP adaptor and the MQTT broker from publish/subscribe design). This is left as future work.

a: INSIGHTS

- Performance is the main focus of networking works. It can have the form of system utilization (CPU, memory, registers, network stack, etc.), throughput, processing power, energy consumption, extendibility and scalability of available resources.
- The crucial point is what components of eBPF are being used and for what purpose. Feedback from eBPF probes can be used to collect information (resource usage, packets information). eBPF maps can be used to store data to be efficiently accessible from userspace or kernel space, or to be a point for exchanging data between them.
- Gathered data are used for profiling (applications, users, devices), modifying packets headers (changing flow direction) copying packets (duplicated for analysis and redundancy) and change connected devices' behaviour based on the collected information (powering off idle devices, controlling network protocols). These can be used for better load balancing, congestion control, reduction of resource usage for overloaded devices, and reduction of energy consumption for idle devices.

Table 1 is a summary of the discussed studies related to the networking domain along with their key aspects, pointing out the limitations of the proposed designs and the future areas left for exploring. The first column refers to the study utilizing eBPF, the second column defines the place in which the eBPF program is stored and executed (Kernel Space (KS), Userspace (US), Network Interface Card (NIC), Storage Service (SS)), the third column defines the main role of the eBPF program, the fourth column summarizes the main aspects of the study, and the fifth column reports the challenges for possible future directions as mentioned by the authors.

B. SECURITY

A vital area where the power of eBPF can be leveraged is data and network security. As attackers can get unprivileged access to resources and data lying in servers, security breaches may lead to data modification or destruction, leak of confidential user or company data and ultimately negative advertising and loss of user trust in the services offered by a company.

The importance of eBPF in security is its highspeed processing that can enable packet analysis before the packet gets into the system and even before reaching the network stack in the kernel. Moreover, eBPF is a robust tool that is used for efficient data gathering when it comes to building training datasets for machine learning-based security applications. It also can provide a level of trustworthiness to the increased untrusted open-source codes since eBPF already has a built-in verifier. The following are the security-related studies we examined in this work.

1) DNS PRIVACY

Domain Name System (DNS), as the root of the world-wide internet, is the basis for controlling the internet flow, node's connectivity, and user interaction with the internet. Gaining access to the DNS layer provides deep insights into user behavior. Distributing single-user traffic across different DNS providers helps maintain users' privacy. In [69] a new

TABLE 1. Summary of eBPF use-cases in the networking domain.

Ref.	Place	Role	Key Aspects	Open Challenges and Future Work
[29]	KS	Storing the value-to-action records	Iptables rules translator to eBPF iptables version	N/A
[30]	KS	Duplicating and redirecting packets to the monitoring VM	Better CPU utilization and higher throughput Extendibility and scalability	XDP integration More testing scenarios
[32]	KS	Data collection of runtime kernel statistics and decisions	Zero impact on system performance High accuracy	Applying deep reinforcement learning techniques for training Hardware resources usage measurement
[35]	KS	Building the data plane for NFs	Higher throughput Better CPU utilization	Analysis of chain NFs and the removal of redundant features Polycube's scalability
[40]	KS	Encoding NFs as eBPF code	Generic version	Overhead and throughput issues
[42], [43]	KS	Collecting real-time data for TCP behavior and network parameters	Higher throughput and consistent data transfer rates with lower packet retransmission rates Real time modification to TCP window size	Investigating more detection perspectives
[49]	KS	Real-time decisions of packet forwarding and cloning policy	Better network and system resource utilization Dynamic network and packet filtering control	N/A
[51]	KS / NIC	Packet queueing and steering	Transparent NIC-CPU scheme to improve request-level parallelism	Performance evaluation
[52], [53]	KS	Network functions creation	Higher packet processing rate Scalability with number of cores No rigid machine resource partitioning	Deep packet inspection and charging Exploiting run-time injection capabilities Studying cross-modules optimization mechanism
[55]	KS	Storing and detecting a link status changes (UP/DOWN) for rerouting	No additional application layer Performance enhancements and robustness of the slave	Overhead reduction Improving performance of the master peer
[58]	KS	Detecting path failure and rerouting Ordering paths and selecting lowest-delay paths	Sender-to-Receiver path independent of Receiver-to-Sender Path ordering/selection based on MAB	Rerouting for better bandwidth utilization Collecting measurement from production traffic
[62]	KS	Event tracing and timing performance	Accurate profiling of execution characteristics of distributed edge applications Targeting user defined events	Profiling real-time applications dynamic configurations of TSN networks Optimizing complex wireless communications with software-based real-time requirements
[44]	KS, NIC	Offloading statistics packets counters and packet-to-application routing tables to the SmartNIC	Traffic accounting, detection of Half-Open TCP connections and DNS flow analysis Offloading IP-to-Application routing tables to NIC Scalability with the amount of resources available	Limitations of SmartNIC XDP offloads Stateful operations offloading to NIC
[45]	KS	Active and dynamic application flow tracing	Dynamic application flow alteration detection and analysis Dropping probe responses within XDP	Scalability with number of destinations and programmability and user control of ELF Noise in the latency measurements
[48]	KS	Monitoring wired-to-wireless packets switching, and tracking energy consumption of APs stations duty-cycle	Reducing energy consumption and number of awakening times No extra hardware equipment needed in the AP stations	Reacting to WiFi network dynamics changes
[64]	KS	Packets IP alteration and packets replication and forwarding	Higher forwarding speed Less overhead	Delivery guarantees (Reliability) Scalability of the eBPF-CB
[65]	KS	Event-based queuing on top of the ethernet interface	Integrating Knative with MQTT XGBoost-based traffic prediction Designing event-based queue Low CPU overhead	Additional Latency of new components Evaluating XGBoost-based prediction component Trying different prediction schemes
[46]	KS	Managing and storing monitored hosts IPs and collect their flow stats counters	Low implantation complexity and CPU overhead Independent of the hosting system architecture	N/A
[60]	KS	Counting number of TCP retransmissions	Low CPU usage Real-time detection of segment losses Not limited to SD-WAN traffic	Limited to TCP traffic Studying other link down causes Missing base reference comparison

scheme was proposed to enhance user privacy by changing the DNS flow for each application running in the system. eBPF provided users with insights over their DNS network traffic to assist in choosing which DNS servers to use and which path to follow. The proposed solution protects users from data leakage and privacy attacks compared to the standard methods.

2) AUDIT LOGGING IN FORENSICS ANALYSIS

Forensics is about analyzing any digital system to discover the attacker's fingerprints as a postponed investigation action. Two critical requirements for forensics analysis include recording system activities and presenting the data concisely. Rohit [70], designed an eBPF-based framework for efficient audit logging that adds a runtime overhead of just 1%.

The audit system's strength is its presentation of the collected data from two primary sources, kernel activities tracked by the default configuration of eBPF and the user's run-time defined instrumentation code. The presented audit system helped in expanding the available auditing tools while overcoming their limitations by reducing the generated audit log size, keeping low run-time overhead, no additional modules to be installed in the kernel, and making it suitable and scalable for use with multiprocessing environments.

3) SECURE AUDITING FOR CONTAINERS

The authors in [71] proposed saBPF a deployable secure auditing tool for containers security. Their design was evaluated against Linux Security Modules (LSM) [72] to ensure that saBPF is maintaining the same performance level as the traditional auditing tools, thus proving its practicality. The authors extended the eBPF framework to allow customizability of audit rule configurations at compilation time so that each container has its own set of configured rules and audit policies. Each individual container can deploy a separate secure auditing tool regardless of other containers in the same hosting system without affecting the host system performance. The authors emphasized the effects of doing the configuration at compilation time instead of run-time, which minimized the run-time audit complexity and improved overall performance.

4) MACHINE LEARNING BASED IDS

Bachl et al. [22] developed an intrusion detection system based on machine learning that runs on the kernel utilizing eBPF. Their solution is using decision trees to process and decide whether a packet is a malicious one while considering the context of the old packets. In order to have accurate results, they implemented their IDS as a userspace application and as an eBPF program. Their results showed an increase of 20% on the number of processed packets for the kernel IDS over the userspace version.

5) CONTAINERS IDS

A good use-case proposed in [73] is to apply eBPF's monitoring power for developing intrusion detection systems specialized in containers security. The paper showed how the technology could retrieve the complex container and application-level context and improve the legacy runtime security tools by reducing the performance impact on the host.

6) HOST-BASED ANOMALY DETECTION SYSTEM

ebpH [16] is an eBPF-based version of Somayaji's pH design [74]. The idea behind both algorithms is to build profiles for each executable on the system. The profiles establish a normal behavior record for the system processes. When a system call violates the established profiles, ebpH will raise a warning to the user and delay that system call so the user can respond and take appropriate action. Performance-wise, ebpH detection produced negligible overhead to the system while maintaining safety guarantees because of eBPF. The code of ebpH is available as open-source, making it a suitable project for individuals to implement a host-based anomaly detection system on their own devices.

7) FAST PATTERN MATCHER BASED IDS

Wang and Chang [24], designed a Snort [75] like IDS based on eBPF. While Snort is designed to run in the user space as an intrusion detection and prevention tool, Wang's and Chang's implementation is running in the kernel. Their approach is based on two newly added components, a control program in the user space to configure the execution of the eBPF program, and an eBPF-based Fast Pattern Matcher (FPM) that runs on the kernel level. The FPM engine is based on the Aho-Corasick (AC) algorithm [76], constructing a state machine from the keywords, and then, in a single pass, figuring out the longest matching pattern. Although subjected to eBPF-limitations, the experimental results demonstrated efficiency of their design against Snort under the same testing conditions.

8) SERVICE MESH OBSERVABILITY IN MICROSERVICES

The evolvement of microservices in containers raised the problem of coordinating the traffic that is directed to the same service. The host device is the coordinator in the standard case which leads to the development of management applications. One of the design patterns is referred to as service mesh (e.g. Istio [77]), illustrated in Figure 14(a). The design of service mesh is to have a layer (the control plane) placed transparently on top of the microservices allowing traffic management, observability, and security, without modifying the service code. The problem of modern tools is the use of static and rigid metrics to collect events and logs. Levin proposed ViberProbe [78] as the first (at the time of writing) scalable eBPF-based dynamic and adaptive microservice metrics collection framework. eBPF will assign application specific configurations and collect their metrics. The collected data can then be used to generate the new configuration based on service pattern analysis and then be deployed and distributed as shown in Figure 14(b).

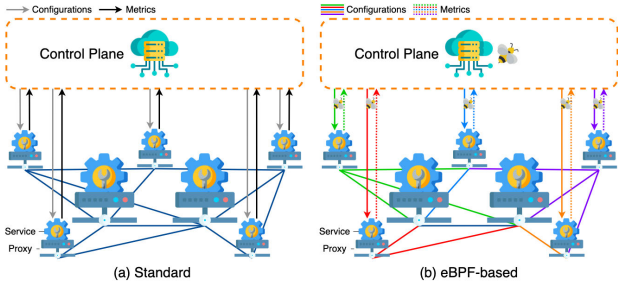


FIGURE 14. Service mesh overview.

9) VM AND CONTAINERS PACKETS VISIBILITY

Deri et al. [79] considered the problem of packet visibility as a result of data encryption while entering/leaving

VMs and containers. The standard method to solve this issue is by deploying packet analyzers per VM or container, which is infeasible in real-life scenarios. The authors designed a novel tool that enables communications visibility at the operating-system level by monitoring network activities using kernel probes and tracepoints rather than normal IDS/IPS which monitors packets crossing the system boundaries.

10) CONTAINERS POLICIES ENFORCEMENT

Containers share host resources, and the way containers interact with the host is by sending system calls (syscalls). Containerizing applications enable the reusability of the host resources to deploy more applications. In this manner, per-container policies are necessary to support different security protocols on the deployed applications in the host system.

Bélair et al. [80] presented SNAPPY which stands for Safe Namespaceable And Programmable PolicY. The SNAPPY framework allowed the definition of per-namespace policies for Linux Security Model (LSM) hooks. The paper's main contributions were allowing untrusted processes to enforce eBPF-based policies into the kernel using the designed namespace policy_NS and implementing dynamic eBPF helpers that can be loaded at runtime. The authors also demonstrated that SNAPPY could be integrated with relevant containers engines (e.g. Docker). Similar work has been done by Findlay et al. [81] as they proposed BPFcontain, a flexible policy language to confine containers and ensure least-privilege access.

11) PROCESS CONFINEMENT

Process confinement is defined by limiting open-source codes and untrusted codes from running and accessing unauthorized system resources and other applications. The confinement process adds more overhead to the system, especially the cloud. The objective is to minimize the overhead at a low rate. Findlay et al. [82] presented a proof-of-concept confinement application called bpfbox based on eBPF. Their prototype showed less overhead compared with AppArmor [83]. Proving the power of bpfbox opened several opportunities for future applications related to systems security.

12) MULTI-LEVEL CONTROL FLOW ATTESTATION

System attestation is the process of ensuring the integrity of a system's configuration and the correctness of system execution behavior. In remote attestation, a third party needs to verify that devices are working and behaving according to pre-defined configurations. An attestation certificate is used as proof of the trustworthiness of the system. Papamartzivanos et al. [84] proposed a hybrid Control Flow Attestation (CFA) framework to overcome the limitations of standard remote attestation solutions, which include i) the need for extra hardware installations and ii) the overhead of precise monitoring and tracing of the system's configuration and operation. The framework is based on multi-level monitoring and reporting using eBPF and Intel Processor Trace technologies (Intel PT) [85]. Although the intention was to develop a pure software-based attestation solution, Intel PT is hardware assumed to already exist in the system. The main idea behind their multi-level design is first to monitor the system using eBPF (the authors call it high-level monitoring) similar to the other use-cases, and if suspicious activity is detected and alerted, then use Intel PT to do in-depth investigations. In other words, eBPF will be used for tracing software commands and execution flow as first level attestation; if it fails, second level forensics analysis is initiated where Intel PT is used to access the programs and data memory to detect exploits. Results showed that eBPF tracing is well suited and does not add to the overall overhead. In contrast, the Intel PT, even on the small scale of targeting specific issues, will add extra overhead, which is why the best and most efficient scalable solution for CFA is to use multi-level tracing and reporting. The overhead of monitoring and the inspection depth is related to the level. In the case of a two-level system, the low level (i.e., first level) has less overhead but at the cost of inspection depth, while the high level (i.e., second level) generates more overhead but with an accurate and precise inspection picture.

13) ASYMMETRIC DoS DETECTION

Denial of service (DoS) attacks aim at interrupting the system and loading it with useless traffic or asking for intensive computations to occupy available resources, thus preventing regular users from accessing system services. Attack mitigation is about reducing the effect of the attack or preventing the attack from happening in the first place. One way of classifying DoS attacks is by observing the number and the type of resources used to launch the attack against the targeted resources [86]. When they are of the same amount and type, it is referred to as Symmetric DoS, and if the resources used are less than the targetted ones or of different types, they are called Asymmetric DoS (ADoS). Classical methods of identifying DoS attacks are no longer applicable in case of ADoS due to its unique characteristics [87].

FINELAME [88] is a monitoring tool with ADoS attacks in mind as its prime focus for detection. Using eBPF, three components have been introduced; i) attaching application-level probes to the functions handling the processing of the incoming requests, ii) monitoring data resources in both kernel/user space, and iii) building a semi-supervised learning model trained on the gathered data to detect anomalies in the requests pattern. FINELAME has reached almost 100% accuracy distinguishing between attacking and legitimate requests while being subjected to ReDoS, Billion Laughs, and SlowLoris attacks.

14) AUTO RULE GENERATION FOR DDoS MITIGATION

Distributed Denial of Service (DDoS) is a category of DoS attacks that uses infected computers called Bots, controlled by the attacker computer (master), to send useless traffic to the victim system to overflow its resources. This results in distributed machines sending malicious traffic without any

tracepoints to investigate the origins of the attack initiator. The attacker starts by sending commands to the controlled servers (CS) which then deliver these commands to all infected computers (called zombies). The infected computers will send a massive amount of useless traffic to the victim, which will use all of its available resources to handle the incoming requests. When an average user sends safe packets to the affected machine, the packets will be dropped due to insufficient resources.

Wieren [89] discussed the use of eBPF for auto rules generation to build DDoS mitigation systems. As illustrated in Figure 15, the legitimate and the invalid traffic will pass into the XDP filter, when the accepted traffic will be objected to an anomaly detection system (ADS). ADS will permit the legitimate traffic to go through and once an anomaly is detected, a copy of the packets will be sent to rules generation systems. A set of new filters will be produced and attached to the XDP filter, and the cycle is repeated. The study demonstrated the potential of getting a filtering accuracy of 100%.

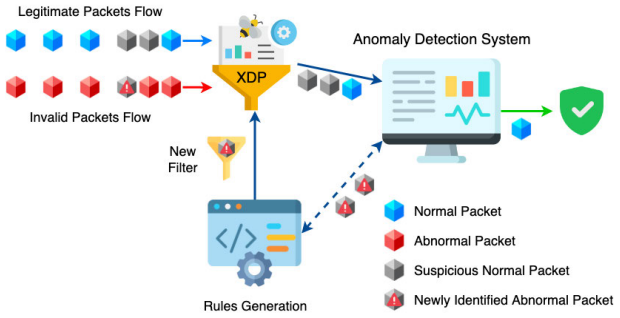


FIGURE 15. Auto rule generation for DDoS attacks.

15) STEGANOGRAPHIC MALWARE DETECTION

Steganography is the process of hiding a message inside a legitimate image, file, or code that is considered not secret. Steganographic malware is a steganographic approach where the embedded information is malware. Caviglione et al. [23] showed the possibility of detecting steganographic malware (stegomalware) using eBPF programs with minimal overhead. Their results showed that the most straightforward eBPF code could provide valuable data to be sent and analyzed by advanced and sophisticated techniques like machine learning-based tools for stegomalware detection. Similarly, Carrega et al. [90] discussed the idea of data gathering using eBPF for building a generalized model for detecting hidden information that could act as a threat.

a: INSIGHTS

- Security is an important requirement of networked systems. It can help protect against unauthorized access and misuse of data or resources by implementing defense mechanisms and controls at various levels of the network and system stack. The use-cases studied here are applying filtering rules and policies on various system levels, changing packets flow, monitoring and coordinating the traffic between separated microservices, and data gathering for building security models.
- Gathered data are used to develop accurate, efficient and trustworthy security tools to detect suspicious packets, irregular and abnormal behavior, profile violations, as well as audit and train machine learning-based security models.
- Filtering rules and policies constructed out of the gathered data can be applied in the kernel, NIC, per-container/VM, in-between containers/VMs, and even on the function-based level (before, after, and inside a function call in the kernel) while maintaining the low overhead profile.
- Monitoring and coordinating traffic for separated microservices (geographically separated applications or servers) helps analyzing the collected data and patterns to push new tuned metrics and configurations.

Table 2 summarizes key aspects of the discussed studies, pointing out the limitations of the proposed designs and the future areas left for exploring.

C. STORAGE

This subsection will summarize studies related to problems with storage devices and the ways required to enhance performance. eBPF can be used as a tool to speedup memory handling or executing operations remotely on data residing in the storage devices.

1) KERNEL-BYPASSING FOR NVMe STORAGE

One of the problems of new Non-Volatile Memory Express (NVMe) storage devices is the overhead produced by the kernel storage path as it is responsible for half of the access latency. Zhong et al. [20] designed an eBPF program that handles the accessing and traversing of dependent requests for data located in storage by applying the concept of kernel-bypass and avoiding the extra delay caused by crossing boundaries between user space and kernel space. Their benchmarks showed that it is possible to reduce latency by half and increase input/output operations per second (IOPS) by 2.5x.

2) EXECUTION OF OPERATIONS ON REMOTE STORAGE

The rapid development and improvements of NVM storage devices technology (e.g. Samsung PM1743 [91] with 13 GB/sec (gigabytes per second) read/write speed) has resulted in the reduction of the memory-to-storage speed gap. On the other hand, network devices (e.g. Ethernet, NIC) are improving at a slower rate. For example a typical Ethernet NIC can handle a traffic of 1 Gb/sec (gigabits per second), which is extremely far from the speed of storage devices (e.g. Samsung 980 Pro with read speed of 7 GB/sec (56 Gb/sec) and write speed of 5 GB/sec (35 Gb/sec)). This performance gap raised the problem of "data movement wall". Kourtis et al. [18] proposed a prototype discussing near-memory computing to support remote operations execution

TABLE 2. Summary of eBPF use-cases in the security domain.

Ref.	Place	Role	Key Aspects	Open Challenges and Future Work
[69]	KS	Finding the amount of leaked information to the DNS and alternating packets DNS destinations	Enforce application specific DNS servers User controllability of DNS network traffic	Further studies on DNS (encrypted traffic, filtration and alteration, queries control) Integrability of DoH and DoT
[70]	KS	Tracking Kernel activities at specific tracepoints	Less overhead Small file sizes generated	Supporting all system calls Selective logging and audit logging mechanisms
[71]	KS	Extending and customizing original eBPF framework with containers-level requirements	Attaching on the intersection between namespaces and reference monitor	cgroup lookup overhead
[22]	KS	Implementing Decision Tree-based Machine Learning model in kernel	Performance increase Higher packet inspection rate	Evaluation against other works Alternative machine learning techniques (RF, DNN)
[73]	KS	Capturing Kernel activities	Defining boolean expressions with limitless complexity File integrity monitoring and suspicious process execution detection	Expanding event types Performance evaluation of host overhead
[16]	KS	Building profiles for executables and detect profiles violations	eBPF version Somayaji's pH	Security analysis, integrability, usability and response automation
[78]	KS	Gathering nodes metrics	Dynamic metrics collection Offline pattern analysis Support horizontal autoscaling	Combined online and offline techniques Effects of unknown patterns Integration with existing microservices management systems
[79]	KS	Capturing the starting parameters and ending values of network events-based functions	Running as an active tool with 0.01% event losses Function level policy enforcements	Handling large number of rules Enabling loops like commands
[80]	KS	Allowing containers to push their own designed policies as eBPF code to the host kernel	Per-container policies enforcements Innovating additional eBPF helpers Additional overhead of 0.1%	Validating the new eBPF helpers Helpers' recursion issue Transformation of LSM modules into SNAPPY
[81]	KS	Enforcing the policy on the confined application	Per-container policies enforcements New YAML-based policy configuration language Adding audit logging capabilities	New eBPF helpers to move process groups into per-container new namespaces The removal of the Daemon Integration with existing container solutions
[82]	KS	Enforcing policies on userspace and kernel space functions	Audit data enabled integrable with audit2allow-like	Unified solution for process confinement for known technologies Extendibility to container level policies
[88]	KS	Data gathering for a semi-supervised learning model of resource utilization	Live detection and resource usage monitoring to user/kernel space Attaching application-level interposition probes to request processing functions	Applying more complex machine learning techniques
[89]	KS	Generating traffic filtering rules based on network packets parameters	Hybrid signature and anomaly-based mitigation system Auto XDP rule generation, injection and updating	Packet filtering algorithms and rules limitations Investigating DDoS protections services using content delivery networks (CDNs)
[23], [90]	KS	Data gathering for colluding applications and covert channels	Colluding application and IP-v6 capable covert channels data gathering enabled	Code inspection and deep inspection of execution patterns Extending approach for other threats
[24]	KS	Storing rules information and actions and applying fast pattern matching	High throughput Fast pattern matching algorithm based on Aho-Corasick algorithm Supporting multi-core execution	Handling large eBPF programs Implementing all Snort features Perfectly utilizing all cores
[84]	KS	Monitoring system configurations and executed software commands	Integrating eBPF with Intel PT Scalability with low overhead and precise monitoring	Limited to systems occupied with Intel PT Investigating pure software-based attestation solutions

on storage using eBPF. The authors focused on the three problems faced by programming storage devices: Any storage extension must balance between safety and expressiveness while running extensions must exhibit minimal overhead and highest possible efficiency. Moreover, the compatibility of the extension to support all domains and applications

must be enforced. Two cases of remote execution have been tested. Remote numerical increment operations reduced the latency to half due to halving the network operations requests. Offloading search operations in data indexed as a binary tree could cut down the latency by 86% on a data size of 1 TB.

3) COMPUTATIONAL STORAGE DEVICE OVER NVMe ZONED NAMESPACES

Computational storage device (CSD) is an alternate approach towards near-memory computing and solving the data movement issue. CSD's main objective is to offer a programmable interface to run user defined programs and operations as close as possible to the storage devices, reducing the data movement cost. One of the proposed designs presented by Lukken et al. [92] is Zoned Computational Storage Device (ZCSD) which is using CSD over NVMe Zoned Namespaces with the help of eBPF running in the userspace. ZCSD is still a work-in-progress. However, the initial prototype, showed the potential of performance gains with low software overhead.

4) USERSPACE SWAPPING

Zhong et al. [19] proposed a new swapping scheme named LightSwap running in the userspace to handle page faults (missing pages) and errors (i.e., fetching page errors, corrupted pages, etc.) to reduce the cost of thread context switching by avoiding the slow kernel data path. LightSwap supports local and remote memory (using NIC) to store the data. The analysis of read/write latency of the kernel and user stacks showed that most of the penalty is caused by the kernel stack which can be up to 10 times slower than the user stack, which is why they suggested moving the swapping from being handled in kernel space to userspace. LightSwap is based on 1) the use of Light Weight Threads (LWT) and integration with the conventional swapping mechanism, 2) eBPF technology to store the default thread context and page fault context (caused by a LWT fault) in the eBPF maps, and 3) a new try-catch exception framework. To deal with page faults, eBPF maps can be used as a shared place to store the contexts (thread, LWT page fault) between the userspace and kernel space. On a page fault, the current LWT will be held (storing its current context), wait until the page is fetched and then resume its execution. LWT fault handling (swapping pages) is done asynchronously so that other LWTs will be scheduled to execute while waiting to resolve the fault. The authors proposed the try-catch exception framework to protect applications from paging errors. Applications use the framework to protect desired code from paging errors and leave the resolution of the error to their customized codes. Compared with Infiniswap [93], results showed that LightSwap can reduce the page fault handling latency by 3-5 times and improve throughput by 40% (depending on the environment setup).

a: INSIGHTS

- eBPF is extended to provide efficient access mechanisms of different types of data structures and storage technologies (SSD, NVMe, etc.) and offload simple commands and operations to be executed directly in the local storage or ship them to remote storages.
- Optimizing the data structure while storing and having clear visibility of blocks and pages in the storage by leveraging eBPF allows the user to access the data efficiently. This reduces data retrieval time and the number of requests for the application to obtain the data. Ultimately, this results in less latency, more access per time unit, and less overhead caused by the storage.
- Encapsulating requests with eBPF headers increases their safety guarantees and ability to be placed and kept between the kernel and the storage. This helps user-defined programs to bypass the kernel and ignore kernel layers overhead for both independent requests and chain of dependent requests.
- Finally, use-cases leverage eBPF to explore an alternative solution to offload commands (single, repeated, chunk, dependant, and independent) and running user-defined functions (using storage APIs) to reduce the latency and number of requests for local storage designs as well for data being stored in a distant place (separated hardware).

Table 3 is a summary of the discussed studies along with their key aspects, limitations of the proposed designs and future areas left for exploring.

D. SANDBOXING

The process of ensuring that a program satisfies deployment constraints is known is Sandboxing. A compiler will re-compile the code to ensure that code meets the environment rules and constraints. In eBPF, the static verification and JIT compilation stages which are responsible for outputting the bytecode as described in Section II, represent the sandboxing process. Sandboxing ensures the efficiency of eBPF execution while producing an optimized bytecode version with formal correctness and safety guarantees. Here we present some of the studies discussing program sandboxing using eBPF.

1) eBPF PROGRAM SYNTHESIS

A recent study by Xu et al. [94] showed the importance and the potentials of optimizing eBPF bytecode. The authors developed K2, a program synthesis optimized eBPF bytecode compiler. Their results showed that it is possible to have bytecode with 6-26% reduced size, 0-4.75% better throughput in terms of packets per second per core, and 1.36-55.03% lower average packet-processing latency. K2 works independently of the eBPF static verifier and there are two distinct sets of checks for the code. K2 compiles the code into different outputs and pass them to eBPF verifier to drop the ones that are rejected by the verifier. Then it checks the performance of the versions that passed the verification and compare them in terms of their latency, size, and throughput, as shown in Figure 16. According to the authors, the more time spent

TABLE 3. Summary of eBPF use-cases in the storage domain.

Ref.	Place	Role	Key Aspects	Open Challenges and Future Work
[20]	KS	Traversing persistent application-defined data structures in the Kernel	New high-level fast storage accessing library New eBPF compatible with NVMe hook	Exploring caching and scheduler policies Safety guarantees of the new NVMe hook
[18]	SS	Shipping the execution of simple codes within the network request contents of remote storage to be directly executed on the storage	Supporting eBPF appcode using NBD protocol Running verification and execution in storage devices	Unsupported loops operations and clang loop unrolling issues Proving total program (program termination) Supporting complicated protocols
[19]	KS	Storing faulting LWT context and restoring the default context	User-space page fault handling mechanism Novel try-catch framework to deal with paging errors Latency improvements and throughput increase	Overcoming storage stack overhead Use of LWT in distributed share memory systems
[92]	US	Support programmability with ZNS SSDs	Low footprint Easy programmability, extensibility, and analysis	Integration with libnvme and zoned-enabled file system for storage Supporting different hardware and additional data structures

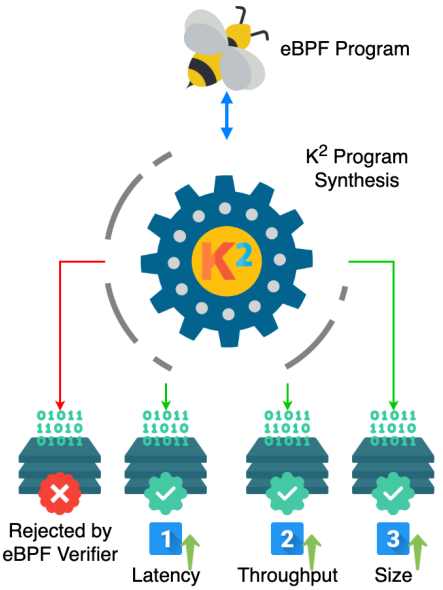


FIGURE 16. eBPF program synthesis.

on the K2 compilation, the more optimized results will be produced.

2) VIRTUAL MACHINES ON IoT MICROCONTROLLERS

The rapid increase of Internet of Things (IoT) devices and microcontrollers with low-power consumption enabled the emergence of dynamic and scalable isolated environments for applications running on microcontrollers. According to Zandberg and Baccelli the requirements of building suitable isolated environments for microcontroller applications as stated in [95] are: memory footprint, extra hardware installation for memory protection, tolerable code execution speed slump, and the amount of data transferred for updating the application. The authors discussed the use of eBPF as a new interesting approach for building tiny VMs scalable and updatable for hosting microcontroller programs. They compared their design rBPF with WASM3 [96] and native C implementation in terms of ROM and RAM usage, code size, and execution times. Initial results showed that rBPF beats both approaches in terms of RAM and ROM usage by 15 orders of magnitude and was capable of executing 1.3 million instructions per second. Although for code size and execution time, rBPF falls behind WASM3 and native C by a large difference, this overhead can be ignored since microcontrollers are not made for computation-intensive use-cases yet.

3) EMBEDDED DEVICES FIRMWARE PATCHING

The generality of eBPF light virtualization and its applicability on different operating systems with low execution requirements (with eBPF there is no need to reboot or restart the system to run an eBPF-based program) made it a perfect solution for running firmware updates to redirect the system flow in case of vulnerabilities existence until proper actions are taken. In [97], a use-case of eBPF was presented towards creating a firmware hot-patching framework for embedded devices. Vendors are facing troubles in patching a variety of embedded devices simultaneously on vulnerabilities or bugs discovery, since they need different firmware configurations and versions. This issue raised the importance of looking for a general firmware patching solution, and here comes the role of eBPF. The authors in [97], developed RapidPatch which is a firmware patching framework capable of generating generic patches for heterogeneous embedded devices to accelerate the patch development and deployment process.

4) HARDWARE-BASED ANDROID NATIVE CODE ANALYZER

Zhou et al. [98] proposed a novel hardware-based Android native code analyzer named NCScope utilizing the built-in features and components of modern Android systems, namely the Embedded Trace Microcell (ETM) [99] and the eBPF. The study presented four use-cases: 1) self-protection (SP)

in financial apps to check the percentage of the apps that implement and secure their execution structure and data, 2) anti-analysis (AA) defense in malicious apps to avoid being detected by software that analyzes their codes, 3) detecting memory corruptions and 4) comparing the performance of functions implemented in the native code. The problems existing solutions face include incomplete instruction traces, high overhead, which can be used as a trigger for the anti-defense mechanisms in malicious apps, and unreliable analysis as a result of testing and analysis performed on emulators rather than in actual systems. ETM is used for instruction traces based on target addresses to track app behavior, while eBPF collects app memory data based on the parameters of the function calls at run-time. Results showed that NCScope introduces a minimal overhead with only a 1.2x slowdown compared to other solutions that can cause up to 40x slowdown to the system. Also, out of 900 financial apps (i.e., banks, wallets, cryptocurrencies, payment apps, etc.), 26.8% implement a SP mechanism in their code, and out of 450 tested malicious apps, 78.3% of them have AA defenses to prevent from being detected. Moreover, the collected data can help diagnose memory bugs and provide helpful and precise insights to evaluate the performance of the native code functions.

a: INSIGHTS

- The key objective in sandboxing studies is to use eBPF virtualization to create more efficient environments (unified, secure) to host re-optimized codes (eBPF bytecode, patches) and analyze programs and their hidden features for a wider range of devices independently of the system specifications and requirements.

Table 4 summarizes sandboxing-related studies.

IV. USERSPACE

Userspace use-cases are very limited in number since most designs focus on processing packets in the earliest possible stage without interrupting any of the system's flows. For example, using XDP, the arrived packets can be processed before even touching the network stack in the kernel. Additionally, XDP can let the packets bypass the kernel to reach userspace for processing. Nevertheless, it might be beneficial to extract and generalize common ideas relevant to userspace processing studies.

In particular, after examining the objectives and the implementation of the reviewed works, we decided to present another grouping approach from the point of view of how these technical solutions interact with eBPF. As shown in Figure 17, we also classified the studies according to their eBPF injection mode, place, type and interfacing with the eBPF maps mode and type. We should stress, however, that this categorization is more restricted in scope as it aims to uncover eBPF usage at the userspace level. Hence, it is not as beneficial as our more general approach of revealing eBPF usage at the Networking, Security, Storage and Sandboxing levels. However, we include it here for completeness.

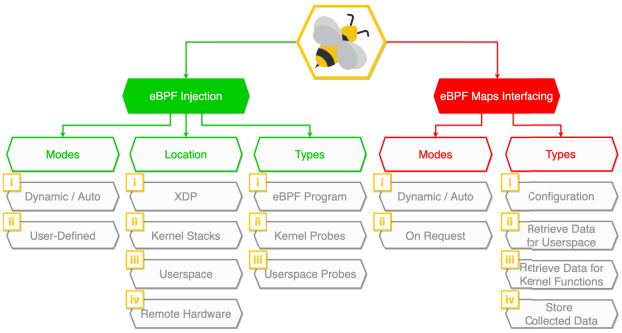


FIGURE 17. Types of userspace relationship to eBPF.

A. eBPF INJECTION

eBPF injection refers to the process of attaching eBPF codes to a system location. Modes of injection can be i) dynamic/auto such that the eBPF program decides when to attach an eBPF code, and ii) user defined since the user decides what to control and attach. From our observations, we concluded that there are four categories for injection locations: i) eXpress Data Path (XDP), since this is the first point of packets leaving the network hardware to interact with kernel space, ii) kernel stacks in the kernel space, iii) userspace, and iv) remote hardware as in case of remote storages. Then we looked at the form of the code and tracepoints that are injected, which we refer to by the term type. We grouped the types into i) eBPF program that refers to custom-defined code to perform a particular task (e.g., filtering and dropping packets), ii) kernel probes (e.g., kprobes and kernel tracepoints) to trace kernel functions, and ii) userspace probes (e.g., uprobes) to trace userspace functions. The following are examples of different eBPF injection modes, location, and type combinations:

1) An eBPF design uses XDP (location) to host auto-generated (mode) filtering rules by analyzing data collected from kernel probes (type).
2) An application developer attaches custom-defined (mode) uprobes (type) in userspace (location) to trace the flow of the application functions in userspace.
3) In data centers, simple instructions (e.g., increment operation) (type) can be shifted to be executed in remote storages (location).

B. eBPF MAPS INTERFACING

The eBPF maps are used mainly for storing data. They have two modes; i) auto/dynamic, where the values stored are updated or retrieved without the user interfering, and ii) on request, such that whatever data the user wants to collect or update is done by triggering the event. As for the types of interfacing, there are four: i) configuring eBPF maps to create and define the needed amount and type of data that must be pre-stored there for later processing, ii) retrieving stored data to userspace to be presented and visualized by the user or further processed in userspace, iii) retrieving stored data for kernel space functions, and iv) storing the collected data

TABLE 4. Summary of eBPF use-cases in the sandboxing domain.

Ref.	Place	Role	Key Aspects	Open Challenges and Future Work
[94]	KS	Executing the old and the new eBPF programs to check their cost functions	Novel synthesis-based eBPF bytecode optimization compiler Adapting stochastics synthesis techniques for BPF New equivalence and safety checker for BPF programs	Scaling to larger programs optimization Proper cost measurement metrics Syncing K2 checker with kernel checker
[95]	KS	Hosting process in an isolated unified environment	Process isolation for low-power microcontroller Better memory usage and utilization	Improving execution time overhead and reducing power consumption and program size Sandboxing guarantees of rBPF
[97]	KS	Distributing and containerizing the firmware patches	Low execution requirements needed Generating device specific code with additional verification step Redirecting execution flow to the desired patch without modifying the ROMs/firmware	Patching large sized features Guaranteeing the patch to be bug-free Distributing the patch on non-connected devices
[98]	KS	Collects and inspect function calls parameters and memory data	Integrating eBPF with ETM Extremely less overhead Undetectable by anti-analysis apps	Integrating with automated testing tools Evaluating more apps Extending the set of SP and AA behavior detection rules

from all eBPF tracepoints types. In a sense, all designs need to deal with the eBPF maps, which means they all have some form of userspace interaction. The following are examples of different interfacing modes and types of combinations:

a) An eBPF program placed in XDP to count different IPs' number of packets and deliver them to an abnormal behavior analysis tool.
b) A TCP monitoring eBPF-based program developed to retransmit packets once they exceed the round trip timeout threshold defined by the network admin stored in the eBPF maps.
c) An optimization tool capturing the traffic of a specific application flow to be used for analyzing its performance with different implementation approaches.

According to what we have seen, all studies are mostly linked to userspace by tracing userspace functions usage, for example, uprobes or fetching the data from the eBPF maps for visualization, further analysis and processing. There have been some studies that specify the userspace scope entirely and differentiate it from kernel space. However, for the majority of works the userspace role is to retrieve data from eBPF maps.

V. FUTURE RESEARCH DIRECTIONS

Due to its key advantages, eBPF has attracted a massive attention both within industry and academia for a broad range of emerging applications. Analyzing a large number of uses-cases of eBPF revealed some of its limitations which create exciting future research opportunities to enhance its capabilities and realize the full potential of this innovative technology.

The eBPF code safety properties are checked and verified by the static analyzer before it can get executed within the kernel. However, the static checker cannot verify even modest size code which limits the complexity of the acceptable programs. Furthermore, generating compact eBPF code for performance enhancement is cumbersome. Formal methods have been on the forefront to address these issues. Nelson et al. in [100] reported an automated proof strategy that can verify the correctness of reasonably large code sizes. Similarly, Vishwanathan et al. in [101] have provided proofs of soundness and optimality for arithmetic operators (addition, subtraction) and proposed a better, precise and faster multiplication algorithm in the domain of tnums (tristate numbers). Improved verification of eBPF programs with soundness guarantees is the key to remove any vulnerability and protect against any kind of attacks. Recently, Demoulin et al. in [94] have proposed a powerful program-synthesis-based compiler that automatically generates compact and fast eBPF bytecode with formal correctness and safety guarantees, which is an important key contribution. However, more follow-up research work is needed in this fertile and important field to verify and optimize large programs quickly with performance and soundness guarantees, and to integrate it with the Linux toolchain to enrich the eBPF capabilities for broader applications such as Femto-Containers [102] for resource constrained low-power IoT devices.

Even though most eBPF works built their grounds on its security guarantees, some vulnerabilities and issues were discovered that need to be addressed before attackers exploit them. Securing eBPF-based applications regardless of the eBPF verification itself is a major concern. These eBPF-based programs introduce new features that may be misused to bypass the verifier and harm the system. SNAPPY [80] when misused, enables a malicious administrator to insert malicious kernel modules leading to a DoS attack. Other works emphasize the role of developers in ensuring that any eBPF-based program is bug-free and works appropriately [97]. Appropriate monitoring of I/O can be

an effective indicator for Cryptolockers (Ransomware) [90]. Draining IoT devices' batteries and DoS attacks can result from missing an execution time boundary of eBPF-based programs [18], [95]. Moreover, the most important aspect about eBPF security is ensuring the correctness and security guarantees of the eBPF verifier itself [71], [81]. Improper updating to the ALU32 bounds used for bitwise operations led to out of bounds arbitrary code execution and therefore, lunching DoS attacks which in turn is used for local privilege escalation (LPE) [103]. Several vulnerabilities concerning the eBPF verifier have been listed in the CVE database [104], pointing to the importance of revisiting the verifier and ensuring its robustness and correctness.

eBPF is a powerful data gathering tool for security analysis and has been used for intrusion detection in several fields such as host-based [16], machine learning [22], and covert channel detection within IPv6 traffic flows [23] with a minimum performance overhead. In [22], the authors used flow-based decision trees to develop a machine learning model which provides a significant performance advantage. An important and interesting direction, ripe for further research, is investigation of other complex machine learning models such as random forest or deep neural networks using eBPF. This will open new avenues for machine learning-based applications of eBPF for load balancing [32], container auditing mechanisms for forensics analysis [71], detecting a wider array of threats such as crypto-lockers and analyzing other performance metrics such as energy consumption, RAM usage patterns, etc. [23].

Networking has been one of the key domains of eBPF applications. Recently, eBPF has been used to enhance the privacy of the standard Domain Name System (DNS), DNS over TLS and DNS over HTTPS communications with low overhead [69], [105]. However, as pointed out by the authors in [69] and [105], more research work is needed to analyze encrypted DNS traffic with the minimum overhead. Similarly, the authors in [42] employed eBPF to get real-time in-band network telemetry information for congestion control in order to greatly increase performance for TCP. This pioneering work opens several exciting avenues for exploring flow control, TCP fairness etc., as reported by the authors [42]. The emerging trend is to use WiFi links for the bulk of Internet traffic. Recent application of eBPF to WiFi access points for determining energy consumption and packet switching delay opens other interesting opportunities for research [48].

Storage latency and I/O operations may become the bottleneck in high performance computer systems. Hence, eBPF has been exploited to reduce the storage access latency in emerging storage devices [20] or reduce performance costs of systems calls [106]. As mentioned in [20] and [106], these interesting works provide more avenues for further exploration such as offloading other key storage operations to the kernel including compression and deduplication. Another promising avenue of programmable computational storage based on non-volatile memory devices is being pursued for in-memory processing (near data processing) to overcome the performance bottleneck in a CPU-driven system for data-intensive workloads [107]. Future extensions are possible to provide for built-in support for popular application level data structures such as B+ trees, hash tables, etc. for near data processing as highlighted by the authors. Another exciting research direction reported by authors in a vision paper [92] is an eBPF-based programmable storage service for the emerging edge computing paradigm. This storage service will allow an application specific customization such as replication, consistency, garbage collection, as well as offloading part of the computation to the storage service in order to meet low latency requirements for a set of disruptive applications like autonomous driving, augmented reality and live video analytics [108].

As cloud services are becoming pervasive, so the data centers are becoming carbon-intensive due to their massive energy consumptions, and hence, attracting global concerns due to their impact on climate. Therefore, reducing the carbon footprint of data centers is an important design goal, which can be addressed both at the hardware and software levels. In the software-centric approach as reported recently in [109], both the energy and carbon footprint need to be made visible to application developers on a fine-grained basis by system API to make informed tradeoffs between performance and carbon emissions. We envision that eBPF can play a vital role in tracking energy and resource usage at a fine-grained level in order to automatically make proper resource selection within the service-level agreements (SLAs) to reduce carbon footprint.

Finally, unified performance and evaluation metrics, and testing policies may be needed to correctly evaluate variety of use-cases for eBPF. eBPF is still under active development and new features are regularly being announced, which will pave the way for new applications to emerge in the near future.

VI. CONCLUSION

The phenomenal growth of cloud services demands new generation of deep monitoring and inspection tools for resource provisioning, traffic engineering, security and stringent QoS/QoE requirements. In this perspective, eBPF has emerged as one of the promising candidate technologies to cope with these challenges in cloud services. Recognizing the pervasive adoption of eBPF in emerging cloud applications, we presented the application landscape of this innovative technology. The prime aim was to offer a use-case oriented comprehensive guidebook of eBPF applications for research and product developers. We started with background knowledge for eBPF, emphasizing its salient features and capabilities. Then, we surveyed four key application domains of eBPF related to networking, security, storage, and sandboxing fields. For each application domain, we discussed and summarized a broad set of use-cases along with their essential characteristics and pros/cons. Finally, we suggested several exciting research opportunities to avail the tremendous potential offered by this technology. In particular, we identified

four promising areas for further exploration which include fast verification and optimization of large eBPF bytecode with performance and soundness guarantees, eBPF-based machine learning applications for security and network management, eBPF-based programmable storage service for the emerging edge computing paradigm, and reducing the carbon footprint of data centers. We hope this study will inspire further developments in the colorful eBPF landscape that will help broaden its scope and usefulness in evolving cloud applications.

REFERENCES

[1] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Gener. Comput. Syst.*, vol. 79, pp. 849–861, Feb. 2018.

[2] A. Bhardwaj and C. R. Krishna, "Virtualization in cloud computing: Moving from hypervisor to containerization—A survey," *Arabian J. Sci. Eng.*, vol. 46, no. 9, pp. 8585–8601, Sep. 2021.

[3] E. Casalicchio and S. Iannucci, "The state-of-the-art in container technologies: Application, orchestration and security," *Concurrency Comput., Pract. Exper.*, vol. 32, no. 17, Sep. 2020, Art. no. e5668.

[4] B. Yi, X. Wang, S. K. Das, K. Li, and M. Huang, "A comprehensive survey of network function virtualization," *Comput. Netw.*, vol. 133, pp. 212–262, Mar. 2018.

[5] A. Kiani and N. Ansari, "Profit maximization for geographically dispersed green data centers," *IEEE Trans. Smart Grid*, vol. 9, no. 2, pp. 703–711, Mar. 2018.

[6] M. Waseem, P. Liang, and M. Shahin, "A systematic mapping study on microservices architecture in DevOps," *J. Syst. Softw.*, vol. 170, Dec. 2020, Art. no. 110798.

[7] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices yesterday, today, and tomorrow," *Present Ulterior Softw. Eng.*, pp. 195–216, 2017, doi: 10.1007/978-3-319-67425-4_12.

[8] M. Waseem, P. Liang, M. Shahin, A. D. Salle, and G. Márquez, "Design, monitoring, and testing of microservices systems: The practitioners' perspective," *J. Syst. Softw.*, vol. 182, Dec. 2021, Art. no. 111061.

[9] S. Yang, F. Li, Z. Zhou, X. Chen, Y. Wang, and X. Fu, "Online control of service function chainings across geo-distributed datacenters," *IEEE Trans. Mobile Comput.*, early access, Dec. 15, 2021, doi: 10.1109/TMC.2021.3135535.

[10] F. K. Parast, C. Sindhav, S. Nikam, H. I. Yekta, K. B. Kent, and S. Hakak, "Cloud computing security A survey of service-based models," *Comput. Secur.*, vol. 114, Mar. 2022, Art. no. 102580.

[11] M. Kleehaus and F. Matthes, "Challenges in documenting microservice-based IT landscape: A survey from an enterprise architecture management perspective," in *Proc. IEEE 23rd Int. Enterprise Distrib. Object Comput. Conf. (EDOC)*, Oct. 2019, pp. 11–20.

[12] S. Karumuri, F. Solleza, S. Zdonik, and N. Tatbul, "Towards observability data management at scale," *ACM SIGMOD Rec.*, vol. 49, no. 4, pp. 18–23, Mar. 2021.

[13] H. Ning, H. Wang, Y. Lin, W. Wang, S. Dhelim, F. Farha, J. Ding, and M. Daneshmand, "A survey on metaverse the state-of-the-art, technologies, applications, and challenges," 2021, *arXiv2111.09673*.

[14] S. McCanne and V. Jacobson, "The BSD packet filter A new architecture for user-level packet capture," in *Proc. USENIX winter*, vol. 46, 1993, pp. 1–11.

[15] *eBPF*. Accessed: Oct. 20, 2021. [Online]. Available: https://ebpf.io/

[16] W. Findlay. (Apr. 2020). *Host-Based Anomaly Detection With Extended BPF*. httpswww.cisl.carleton.ca~willwrittencourseworkundergrad-ebpH-thesis.pdf

[17] N. Hedam, "EBPF-from a programmer's perspective," EasyChair, London, U.K., Tech. Rep. 5198, 2021.

[18] K. Kourtis, A. Trivedi, and N. Ioannou, "Safe and efficient remote application code execution on disaggregated NVM storage with eBPF," 2020, *arXiv:2002.11528*.

[19] K. Zhong, W. Cui, Y. Lu, Q. Liu, X. Yan, Q. Yuan, S. Luo, and K. Huang, "Revisiting swapping in user-space with lightweight threading," 2021, *arXiv:2107.13848*.

[20] Y. Zhong, H. Wang, Y. J. Wu, A. Cidon, R. Stutsman, A. Tai, and J. Yang, "BPF for storage: An exokernel-inspired approach," in *Proc. Workshop Hot Topics Operating Syst.*, Jun. 2021, pp. 128–135.

[21] M. A. M. Vieira, M. S. Castanho, R. D. G. Pacífico, E. R. S. Santos, E. P. M. C. Júnior, and L. F. M. Vieira, "Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications," *ACM Comput. Surv.*, vol. 53, no. 1, pp. 1–36, Jan. 2021.

[22] M. Bachl, J. Fabini, and T. Zseby, "A flow-based IDS using machine learning in eBPF," 2021, *arXiv:2102.09980*.

[23] L. Caviglione, W. Mazurczyk, M. Repetto, A. Schaffhauser, and M. Zuppelli, "Kernel-level tracing for detecting stegomalware and covert channels in Linux environments," *Comput. Netw.*, vol. 191, May 2021, Art. no. 108010.

[24] S.-Y. Wang and J.-C. Chang, "Design and implementation of an intrusion detection system by using extended BPF in the Linux kernel," *J. Netw. Comput. Appl.*, vol. 198, Feb. 2022, Art. no. 103283.

[25] *Sunos 4.1.1 Reference Manual*, S. M. Inc, Mountain View, CA, USA, Oct. 1990.

[26] BIBentryALTinterwordspacing T. Hoiland-Jorgensen, 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 *Proc. 14th Int. Conf. Emerg. Netw. EXperiments Technol.*, New York, NY, USA: Association for Computing Machinery, 2018, pp. 54–66, doi: 10.11453281411.3281443.

[27] J. Evans, "Linux tracing systems & how they fit together," Tech. Rep., 2017. [Online]. Available: https://jvns.ca/blog/2017/07/05/linux-tracing-systems/

[28] *Kernel Development Community*. Accessed: Oct. 23, 2021. [Online]. Available: httpswww.kernel.org

[29] A. Deepak, R. Huang, and P. Mehra, "eBPFXDP based firewall and packet filtering," in *Proc. Linux Plumbers Conf.*, 2018, pp. 1–5.

[30] J. Hong, S. Jeong, J.-H. Yoo, and J. W.-K. Hong, "Design and implementation of eBPF-based virtual tap for inter-VM traffic monitoring," in *Proc. 14th Int. Conf. Netw. Service Manage. (CNSM)*, 2018, pp. 402–407.

[31] *Open Vswitch*. Accessed: Oct. 25, 2021. [Online]. Available: httpwww.openvswitch.org

[32] J. Chen, S. S. Banerjee, Z. T. Kalbarczyk, and R. K. Iyer, "Machine learning for load balancing in the Linux kernel," in *Proc. 11th ACM SIGOPS Asia–Pacific Workshop Syst.*, Aug. 2020, pp. 67–74.

[33] M. Chiosi et al., "Network functions virtualisation An introduction, benefits, enablers, challenges and call for action," in *Proc. SDN OpenFlow World Congr.*, vol. 48, 2012, pp. 1–16.

[34] *European Telecommunications Standards Institute (ETSI)*. Accessed: Nov. 3, 2021. [Online]. Available: httpswww.etsi.orgtechnologiesnfv

[35] S. Miano, F. Risso, M. V. Bernal, M. Bertrone, and Y. Lu, "A framework for eBPF-based network functions in an era of microservices," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 133–151, Mar. 2021.

[36] *Data Plane Development Kit*. Accessed: Jul. 10, 2022. [Online]. Available: httpswww.dpdk.org

[37] L. Rizzo, "Netmap a novel framework for fast packet IO," in *Proc. 21st USENIX Secur. Symp. (USENIX Secur.)*, 2012, pp. 101–112.

[38] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vSwitch," in *Proc. 12th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, May 2015, pp. 117–130. [Online]. Available: httpswww.usenix.orgconferencensdi15technical-sessionspresentationpfaff

[39] C. Systems. *Segment Routing*. Accessed: Mar. 7, 2022. [Online]. Available: httpswww.segment-routing.net

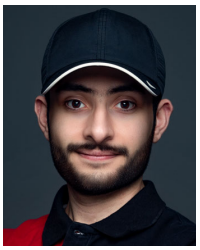
[40] BIBentryALTinterwordspacing M. Xhonneux, F. Duchene, and O. Bonaventure, "Leveraging ebpf for programmable network functions with ipv6 segment routing," in *Proc. 14th Int. Conf. Emerg. Netw. Exp. Technol.*, New York, NY, USA: Association for Computing Machinery, 2018, pp. 67–72, doi: 10.11453281411.3281426.

[41] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li, "In-band network telemetry: A survey," *Comput. Netw.*, vol. 186, Feb. 2021, Art. no. 107763. [Online]. Available: httpswww.sciencedirect.comsciencearticlepiiS1389128620313396

[42] R. V. Bhat, J. Haxhibeqiri, I. Moerman, and J. Hoebeke, "Adaptive transport layer protocols using in-band network telemetry and eBPF," in *Proc. 17th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2021, pp. 241–246.

[43] X. Dong and Z. Liu, "Multi-dimensional detection of Linux network congestion based on eBPF," in *Proc. 14th Int. Conf. Measuring Technol. Mechatronics Autom. (ICMTMA)*, Jan. 2022, pp. 925–930.
[44] M. Abranches, O. Michel, E. Keller, and S. Schmid, "Efficient network monitoring applications in the kernel with eBPF and XDP," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2021, pp. 28–34.
[45] J. Sommers and R. Durairajan, "Elf High-performance in-band network measurement," in *Proc. TMA*, 2021. [Online]. Available: https://ix.cs.uoregon.edu/~ram/papers/TMA-2021.pdf
[46] Z. Zha, A. Wang, Y. Guo, and S. Chen, "Towards software defined measurement in data centers: A comparative study of designs, implementation, and evaluation," *IEEE Trans. Cloud Comput.*, early access, Jun. 10, 2022, doi: 10.1109/TCC.2022.3181890.
[47] A. Wang, Y. Guo, F. Hao, T. V. Lakshman, and S. Chen, "Umon Flexible and fine grained traffic monitoring in open vswitch," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol.*, 2015, pp. 1–7, doi: 10.11452716281.2836100.
[48] J. Sheth, V. Ramanna, and B. Dezfouli, "FLIP: A framework for leveraging eBPF to augment WiFi access points and investigate network performance," in *Proc. 19th ACM Int. Symp. Mobility Manage. Wireless Access*, Nov. 2021, pp. 117–125.
[49] S. Baidya, Y. Chen, and M. Levorato, "EBPF-based content and computation-aware communication for real-time edge computing," 2018, *arXiv:1805.02797*.
[50] A. Kaufmann, S. Peter, N. K. Sharma, T. Anderson, and A. Krishnamurthy, "High performance packet processing with flexnic," in *Proc. 21st Int. Conf. Architectural Support Program. Lang. Operating Syst.*, New York, NY, USA: Association for Computing Machinery, 2016, pp. 67–81, doi: 10.11452872362.2872367.
[51] P. Enberg, A. Rao, and S. Tarkoma, "Partition-aware packet steering using XDP and eBPF for improving application-level parallelism," in *Proc. 1st ACM CoNEXT Workshop Emerg. Netw. Comput. Paradigms*, 2019, pp. 27–33, doi: 10.11453359993.3366766.
[52] F. Parola, "Prototyping an eBPF-based 5g mobile gateway," M.S. thesis, Dept. Comput. Eng., Politecnico di Torino, Turin, Italy, 2020.
[53] F. Parola, F. Risso, and S. Miano, "Providing telco-oriented network services with eBPF: The case for a 5G mobile gateway," in *Proc. IEEE 7th Int. Conf. Netw. Softwarization (NetSoft)*, Jun. 2021, pp. 221–225.
[54] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2014, pp. 1–6.
[55] M. Xhonneux and O. Bonaventure, "Flexible failure detection and fast reroute using eBPF and SRV6," in *Proc. 14th Int. Conf. Netw. Service Manage. (CNSM)*, 2018, pp. 408–413.
[56] A. Bashandy, C. Filsfils, B. Decraene, S. Litkowski, P. Francois, D. Voyer, F. Clad, and P. Camarillo. (Oct. 2018). *Topology Independent Fast Reroute Using Segment Routing*. Internet Engineering Task Force, Internet-Draft Draft-Bashandy-RTGWG-Segment-Routing-TI-LFA-05. [Online]. Available: httpsdatatracker.ietf.orgdochtmldraft-bashandy-rtgwg-segment-routing-ti-lfa-0
[57] D. Katz and D. Ward, *Bidirectional Forwarding Detection (BFD)*, document RFC 5880, Jun. 2010. [Online]. Available: httpsrfc-editor.orgrfcrfc5880.txt
[58] M. Jadin, Q. De Coninck, L. Navarre, M. Schapira, and O. Bonaventure, "Leveraging eBPF to make TCP path-aware," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 3, pp. 2827–2838, Sep. 2022.
[59] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," 2012, *arXiv:1204.5721*.
[60] S. Troia, M. Mazzara, M. Savi, L. M. M. Zorello, and G. Maier, "Resilience of delay-sensitive services with transport-layer monitoring in SD-WAN," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 3, pp. 2652–2663, Sep. 2022.
[61] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "Onos towards an open, distributed sdn os," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 1–6, doi: 10.11452620728.2620744.
[62] V. Gowtham, O. Keil, A. Yeole, F. Schreiner, S. Tschoke, and A. Willner, "Determining edge node real-time capabilities," in *Proc. IEEE/ACM 25th Int. Symp. Distrib. Simul. Real Time Appl. (DS-RT)*, Sep. 2021, pp. 1–9.
[63] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publishsubscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003, doi: 10.1145857076.857078.
[64] M. Tatarski, H. Parzyjegla, P. Danielis, and G. Muhl, "Fast publish-subscribe using Linux eBPF," Tech. Rep., 2022. [Online]. Available: http://dx.doi.org/10.15496/publikation-67449
[65] I.-C. Wang, S. Qi, E. Liri, and K. K. Ramakrishnan, "Towards a proactive lightweight serverless edge cloud for Internet-of-Things applications," in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*, Oct. 2021, pp. 1–4.
[66] *Knative Framework*. Accessed: Dec. 5, 2021. [Online]. Available: http-sknative.devdocs
[67] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi, "Internet of Things: Survey and open issues of MQTT protocol," in *Proc. Int. Conf. Eng. MIS (ICEMIS)*, May 2017, pp. 1–6.
[68] T. Chen and C. Guestrin, "Xgboost A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 785–794, doi: 10.11452939672.2939785.
[69] S. Rivera, V. K. Gurbani, S. Lagraa, A. K. Iannillo, and R. State, "Leveraging eBPF to preserve user privacy for DNS, DoT, and DoH queries," in *Proc. 15th Int. Conf. Availability, Rel. Secur.*, Aug. 2020, pp. 1–10.
[70] R. Aich, "Efficient audit data collection for Linux," M.S. thesis, Dept. Comput. Sci., Stony Brook Univ., New York, NY, USA, 2021.
[71] S. Y. Lim, B. Stelea, X. Han, and T. Pasquier, "Secure namespaced kernel audit for containers," in *Proc. ACM Symp. Cloud Comput.*, Nov. 2021, pp. 518–532.
[72] BIBentryALTinterwordspacing C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux security modules General security support for the Linux kernel," Linux security modules: General security support for the Linux kernel," in *Proc. Found. Intrusion Tolerant Syst., [Organically Assured Survivable Inf. Syst.]*, 2002. [Online]. Available: httpswww.usenix.orgconference11th-usenix-security-symposiumlinux-security-modules-general-security-support-linux
[73] G. Fournier, S. Afchain, and S. Baubeau, "Runtime security monitoring with eBPF," in *Proc. 17th Symp. la Sécurité des Technol. del'Inf. et de la Commun. (SSTIC)*, 2021, pp. 1–23.
[74] A. B. Somayaji, "Operating system stability and security through process homeostasis," Univ. New Mexico, Albuquerque, NM, USA, 2002.
[75] *Cisco*. Accessed: Dec. 10, 2021. [Online]. Available: httpswww.snort.org
[76] A. V. Aho and M. J. Corasick, "Efficient string matching An aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, Jun. 1975, doi: 10.1145360825.360855.
[77] Istio. *The Istio Service Mesh*. Accessed: Dec. 15, 2021. [Online]. Available: httpsistio.iolatestaboutservice-mesh
[78] J. Levin and T. A. Benson, "ViperProbe: Rethinking microservice observability with eBPF," in *Proc. IEEE 9th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2020, pp. 1–8.
[79] L. Deri, S. Sabella, and S. Mainardi, "Combining system visibility and security using eBPF," in *Proc. 3rd Italian Conf. Cyber Secur.*, in CEUR Workshop Proceedings, vol. 2315, P. Degano and R. Zunino, Eds. Pisa, Italy: CEUR-WS.org, Feb. 2019.
[80] M. B'elair, S. Laniepce, and J.-M. Menaud, *SNAPPY Program. Kernel-Level Policies for Containers*.hskip 1em plus 0.5em minus 0.4emrelax New York, NY, USA Association for Computing Machinery, 2021, pp. 1636–1645, doi: 10.11453412841.3442037.
[81] W. Findlay, D. Barrera, and A. Somayaji, "BPFContain: Fixing the soft underbelly of container security," 2021, *arXiv:2102.06972*.
[82] W. Findlay, A. Somayaji, and D. Barrera, "BPFbox Simple precise process confinement with eBPF," in *Proc. ACM SIGSAC Conf. Cloud Comput. Secur. Workshop*, 2020, pp. 91–103, doi: 10.11453411495.3421358.
[83] C. Cowan, S. Beattie, G. Kroah-Hartman, C. Pu, P. Wagle, and V. Gligor, "SubDomain parsimonious server security," in *Proc. 14th Syst. Admin. Conf. (LISA 2000)*. Dec. 2000, pp. 1–20. [Online]. Available: httpswww.usenix.orgconferencelisa-2000subdomain-parsimonious-server-security
[84] D. Papamartzivanos, S. A. Menesidou, P. Gouvas, and T. Giannetsos, "Towards efficient control-flow attestation with software-assisted multi-level execution tracing," in *Proc. IEEE Int. Medit. Conf. Commun. Netw. (MeditCom)*, Sep. 2021, pp. 512–518.
[85] *Intel Platform Analysis Technology*. Accessed: Jan. 5, 2022. [Online]. Available: httpswww.intel.comcontentwwwusendevelopertopic-technologyplatform-analysis-technologyoverview.html
[86] A. Chen, A. Sriraman, T. Vaidya, Y. Zhang, A. Haeberlen, B. T. Loo, L. T. X. Phan, M. Sherr, C. Shields, and W. Zhou, "Dispersing asymmetric ddos attacks with splitstack," in *Proc. 15th ACM Workshop Hot Topics Netw.*, 2016, pp. 197–203, doi: 10.11453005745.3005773.

[87] A. Praseed and P. S. Thilagam, "Modelling behavioural dynamics for asymmetric application layer DDoS detection," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 617–626, 2021.
[88] H. M. Demoulin, I. Pedisich, N. Vasilakis, V. Liu, B. T. Loo, and L. T. X. Phan, "Detecting asymmetric application-layer Denial-of-Service attacks in-flight with FineLame," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, Jul. 2019, pp. 693–708. [Online]. Available: https-swww.usenix.orgconferenceatc19presentationdemoulin
[89] H. V. Wieren. (Nov. 2019). *Signature-Based DDoS Attack Mitigation Automated Generating Rules for Extended Berkeley Packet Filter and Express Data Path*. [Online]. Available: httpessay.utwente.nl80125
[90] A. Carrega, L. Caviglione, M. Repetto, and M. Zuppelli, "Programmable data gathering for detecting stegomalware," in *Proc. 6th IEEE Conf. Netw. Softwarization (NetSoft)*, Jun. 2020, pp. 422–429.
[91] *SAMSUNG*. Accessed: Jan. 18, 2022. [Online]. Available: httpswww.samsung.comsemiconductorssdpm1743
[92] C. Lukken, G. Frascaria, and A. Trivedi, "ZCSD: A computational storage device over zoned namespaces (ZNS) SSDs," 2021, *arXiv:2112.00142*.
[93] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with Infiniswap," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*. Boston, MA, USA: USENIX Association, Mar. 2017, pp. 649–667. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu
[94] Q. Xu, M. D. Wong, T. Wagle, S. Narayana, and A. Sivaraman, "Synthesizing safe and efficient kernel extensions for packet processing," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 50–64.
[95] K. Zandberg and E. Baccelli, "Minimal virtual machines on IoT microcontrollers: The case of Berkeley packet filters with RBPF," in *Proc. 9th IFIP Int. Conf. Perform. Eval. Modeling Wireless Netw. (PEMWN)*. IEEE, 2020, pp. 1–6.
[96] E. Ronen and A. Shamir, "Extended functionality attacks on IoT devices: The case of smart lights," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 3–12.
[97] M. Salehi and K. Pattabiraman, "Poster AutoPatch: Automatic hotpatching of real-time embedded devices," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 3451–3453. [Online]. Available: httpswww.usenix.orgconferenceusenixsecurity22presentationhe-yi
[98] H. Zhou, S. Wu, X. Luo, T. Wang, Y. Zhou, C. Zhang, and H. Cai, "Ncscope Hardware-assisted analyzer for native code in Android apps," in *Proc. 31st ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, 2022, pp. 629–641, doi: 10.11453533767.3534410.
[99] *Embedded Trace Macrocell Architecture Specification*. Accessed: Jan. 28, 2022. [Online]. Available: httpsdeveloper.arm.comdocumentationihi0014q
[100] L. Nelson, J. Van Geffen, E. Torlak, and X. Wang, "Specification and verification in the field Applying formal methods to BPF just-in-time compilers in the Linux kernel," in *Proc. 14th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2020, pp. 41–61.
[101] H. Vishwanathan, M. Shachnai, S. Narayana, and S. Nagarakatte, "Semantics, verification, and efficient implementations for tristate numbers," 2021, *arXiv:2105.05398*.
[102] K. Zandberg and E. Baccelli, "Femto-containers Devops on microcontrollers with lightweight virtualization & isolation for iot software modules," 2021, *arXiv:2106.12553*.
[103] V. Palmiotti. (Mar. 2022). *Kernel Pwning With eBPF a Love Story*. [Online]. Available: httpswww.graplsecurity.compostkernel-pwning-with-ebpf-a-love-story#toc-4
[104] *Common Vulnerabilities and Exposures*. Accessed: Feb. 6, 2022. [Online]. Available: httpscve.mitre.orgcgi-bincvekey.cgikeyword=eBPF
[105] J. Steadman and S. Scott-Hayward, "DNSxP: Enhancing data exfiltration protection through data plane programmability," *Comput. Netw.*, vol. 195, Aug. 2021, Art. no. 108174.
[106] L. Gerhorst, B. Herzog, S. Reif, W. Schröder-Preikschat, and T. Hönig, "AnyCall: Fast and flexible system-call aggregation," in *Proc. 11th Workshop Program. Lang. Operating Syst.*, Oct. 2021, pp. 1–8.
[107] G. Frascaria, A. Trivedi, and L. Wang, "A case for a programmable edge storage middleware," 2021, *arXiv:2111.14720*.
[108] C. Lukken and A. Trivedi, "Past, present and future of computational storage A survey," 2021, *arXiv:2112.09691*.
[109] T. Anderson, A. Belay, M. Chowdhury, A. Cidon, and I. Zhang, "Treehouse A case for carbon-aware datacenter software," 2022, *arXiv:2201.02120*.



HUSAIN SHARAF received the B.Sc. degree in computer engineering from Kuwait University, in 2019, where he is currently pursuing the M.Sc. degree in computer engineering. He is also working as a Cyber Security Engineer at the Electronic and Cyber Crime Combating Department, Criminal Investigation General Department, Ministry of Interior (Kuwait). Prior to that, he was working as a Mobile Application Developer and recently shifted to desktop and web application development as a Freelance Full-Stack Developer. His research interests include machine learning, parallel and distributed computing, cryptography, and quantum computing.

IMTIAZ AHMAD received the B.Sc. degree in electrical engineering from the University of Engineering and Technology Lahore, Pakistan, in 1984, the M.Sc. degree in electrical engineering from the King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, in 1988, and the Ph.D. degree in computer engineering from Syracuse University, Syracuse, NY, USA, in 1992. Since 1992, he has been with the Department of Computer Engineering, Kuwait University, Kuwait, where he is currently a Professor. His research interests include the design automation of digital systems, parallel and distributed computing, machine learning, and software-defined networks.

TASSOS DIMITRIOU (Senior Member, IEEE) is currently a Professor with the Computer Engineering Department, Kuwait University. Prior to that, he was an Associate Professor with the Athens Information Technology, Greece, where he was leading the Algorithms and Security Group, and an Adjunct Professor with Carnegie Mellon University, Pittsburgh, PA, USA, and Aalborg University, Aalborg, Denmark. He conducts research in areas spanning from the theoretical foundations of cryptography to the design and implementation of leading edge efficient and secure communication protocols. Emphasis is given in authentication and privacy issues for various types of networks (ad hoc, sensor and ubiquitous networks, RFID, and smart grid), security architectures for wireless and telecommunication networks, and the development of secure applications for networking and electronic commerce. His research in the above fields has resulted in numerous publications, some of which received distinction, and numerous invitations for talks in prestigious conferences.

He is a member of ACM, a Fulbright Fellow, and a Distinguished Lecturer of ACM. For more information visit the link (http://tassosdimitriou.com/).