SBRC'24

42º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos

20 A 24 MAIO NITERÓI

MINICURSOS

SBC
Sociedade Brasileira de Computação

SBRC'24
42º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos

20 a 24 maio NITERÓI

Organizadores

Antonio A. de Aragão Rocha, UFF
Carlos Alberto Vieira Campos, UNIRIO
Diego Passos, ISEL
Daniel S. Menasché, UFRJ

Antônio Jorge Gomes Abelém, UFPA
Ana Paula Couto da Silva, UFMG

Minicursos do 42º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos

Porto Alegre
Sociedade Brasileira de Computação – SBC
2024

S612m

Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (42. : 2024 : Niterói, RJ)
Minicursos do 42º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos [recurso eletrônico] / Organizadores: Antonio Augusto de Aragão Rocha, Carlos Alberto Vieira Campos, Diego Passos, Daniel Sadoc Menasché, Antônio Jorge Gomes Abelém, Ana Paula Couto da Silva. – Porto Alegre: Sociedade Brasileira de Computação, 2024.

v, 276p. : il.

Inclui bibliografia.
ISBN: 978-85-7669-608-7

1. Informática na educação - Congressos. 2. Sistemas operacionais distribuídos (Computadores) - Congressos. I. Rocha, Antonio Augusto de Aragão. II. Campos, Carlos Alberto Vieira. III. Passos, Diego. IV. Menasché, Daniel Sadoc. V. Abelém, Antônio Jorge Gomes.VI. Silva, Ana Paula Couto da. VII. Sociedade Brasileira de Computação. VIII. Título.

Ficha catalográfica elaborada por Tatiana de Sousa Ribeiro - CRB-7/6199

Sociedade Brasileira de Computação – SBC

Contato
Av. Bento Gonçalves, 9500
Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia
91.509-900 – Porto Alegre RS

CNPJ: 29.532.264/0001-78
http://www.sbrc.org.br

Mensagem dos Coordenadores de Minicursos do SBRC 2024

É com grande entusiasmo que compartilhamos os Minicursos do 42º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), realizado de 20 a 24 de maio de 2024, em Niterói-RJ. Os Minicursos do SBRC contribuem para o Livro de Minicursos, uma publicação reconhecida por seu material de alta qualidade que aborda temas relevantes de pesquisa e desenvolvimento. A forte participação e o nível técnico demonstrado pelos presentes ressaltam a importância desse evento, já consolidado ao longo das edições do SBRC. Tanto o livro quanto as apresentações são concebidos de forma didática, destinados a motivar estudantes e profissionais a se aprofundarem em temas que frequentemente não são abordados nas grades curriculares convencionais.

Neste ano, recebemos 9 propostas de minicursos, todas avaliadas por pelo menos quatro revisores. A organização dos Minicursos do SBRC 2024 expressa gratidão aos 20 membros do comitê de avaliação pelo empenho e dedicação na revisão das propostas. O número de submissões permitiu à coordenação selecionar as 6 propostas mais destacadas. Estas propostas prometem atrair um amplo público, abordando temas como Placas de Redes Inteligentes, WebAssembly, Orientação de Conteúdo para Streaming de Vídeos Adaptativos, Pesquisa Experimental em Cibersegurança, Desaprendizado de Máquinas associado à LGPD e Bases de Dados Sintéticas usando Redes Adversariais. É importante mencionar que algumas propostas não selecionadas teriam plena capacidade de serem aceitas, mas infelizmente não foi possível incluí-las nesta ocasião.

Agradecemos aos organizadores gerais do SBRC 2024, Antônio Augusto Rocha (UFF), Carlos Alberto Vieira Campos (UNIRIO), Daniel Sadoc Menasché (UFRJ) e Diego Passos (ISEL/UFF), pelo compromisso e dedicação na organização do SBRC 2024. Agradecemos também aos autores dos minicursos pelo empenho e pontualidade na entrega do material, essenciais para o sucesso do evento. Estamos confiantes de que as apresentações foram enriquecedoras e contribuíram significativamente para o conhecimento técnico dos participantes. O encerramento dos minicursos foi bem-sucedido graças à colaboração e ao profissionalismo de todos. Agora, esperamos que o trabalho árduo realizado durante o evento estimule discussões frutíferas em prol do avanço da ciência nacional.

Ana Paula Couto da Silva, UFMG

Antônio Jorge Gomes Abelém, UFPA

Comitê de Avaliação de Minicursos

Alberto Egon Schaeffer-Filho, UFRGS
Alex Borges Vieira, UFJF
Alfredo Goldman, USP
Anelise Munaretto, UTFPR
Christian Esteve Rothenberg, UNICAMP
Daniel Ludovico Guidoni, UFOP
Debora Muchaluat-Saade, UFF
Denis Rosário, UFPA
Dianne de Medeiros, UFF
Edmundo Madeira, UNICAMP
Eduardo Coelho Cerqueira, UFPA
Helder Oliveira, UFABC
Igor Monteiro Moraes, UFF
Jussara Almeida, UFMG
Leobino Nascimento Sampaio, UFBA
Mauro Fonseca, UTFPR
Rafael Lopes Gomes, UECE
Rodrigo de Souza Couto, UFRJ
Ronaldo Alves Ferreira, UGMS
Weverton Luis da Costa Cordeiro, UFRGS

Créditos

Arte capa: Juarez Castro, ACS/NCE
Foto capa: Antonio A. de Aragão Rocha, UFF

Sumário

Mensagens dos organizadores iv

Comitês v

1 Content Steering: Leveraging the Computing Continuum to Support Adaptive Video Streaming 1

2 SmartNICs: The Next Leap in Networking 40

3 Generation of Synthetic Datasets in the Context of Computer Networks using Generative Adversarial Networks 90

4 Desaprendizado de Máquinas e a LGPD: da privacidade ao direito ao esquecimento 143

5 Testbeds para Pesquisa Experimental em Cibersegurança: Da Teoria à Prática 186

6 WebAssembly: Uma Introdução 228

Capítulo 1

Content Steering: Leveraging the Computing Continuum to Support Adaptive Video Streaming

Roberto Rodrigues-Filho (UFSC), Eduardo S. Gama (UNICAMP), Marcio Miranda Assis (UNICAMP), Roger Immich (UFRN), Edmundo Madeira (UNICAMP), Luiz F. Bittencourt (UNICAMP)

Abstract

Video streaming has become one of the most used Internet applications nowadays, with numerous leading technology companies competing for dominance in a market valued in the billions. The delivery of high-quality streaming services requires strategic utilization of computing resources near end-users. Emerging technologies like 6G and edge-cloud continuum infrastructures are being explored to meet the growing demands. These technologies promise to enable fast and reliable data transfer for large data volumes, with the edge-cloud continuum facilitating service placement mobility from central cloud data centers to edge devices near users. However, managing seamless service mobility and precise computing resource allocation for quality services remains complex. In the light of this, the zero-touch network concept was born. It eliminates the need for manual network configuration and it is becoming popular in this context. Specifically, in video streaming, the integration of the Content Steering architecture from the Dynamic Adaptive Streaming over HTTP (DASH) protocol with container orchestration technologies could allow for autonomous video streaming service placement across the computing continuum, reducing human involvement and optimizing computing resource use. This short course provides a hands-on experience with the latest technology in this domain, teaching participants about cutting-edge architectures and tools for creating and managing adaptive video streaming applications using the latest content steering architecture introduced in the DASH protocol. Participants will build a small edge-cloud virtual and local testbed to explore request steering strategies for video content across the computing continuum. The course also addresses current challenges and future research opportunities in this evolving field.

1.1. Introduction

Video streaming content is one of the most popular forms of content on the Internet nowadays, with big techs fighting for a market share in the growing video streaming application market. This scenario has motivated the research and development of new techniques and methods for supporting the demands for video content streaming in today's networking infrastructures.

Over the years, many technologies have been developed to better support the growing demands for video streaming content [51, 33, 53]. 5G networks and the expected 6G networks are recent technologies supporting fast access to this type of content [66]. Furthermore, Zero-touch Network and Service Management [40] is being investigated to provide closed-loop autonomous control over the configuration and optimization of such infrastructures and enable them to support resource-intensive applications such as video streaming.

Aligned with these technological advances to create and manage better infrastructures to support the demands of contemporary services, the Edge-cloud Continuum is an emerging infrastructure that has gained popularity in recent years. These are hierarchical infrastructures of highly heterogeneous devices that comprehend the end-user and IoT devices (e.g., laptops, smartphones, drones, sensors, actuators, etc.), a supporting network of edge/fog computing that brings the capability of cloud services closer to end users and extends itself to cloud datacenters [10].

This resulting computing infrastructure enables and supports a plethora of applications that generate and process high volumes of data. It enables the strategic deployment and allocation of video streaming services, bringing them closer to end-users and thereby minimizing network latency, as noted by Gama et al. [27]. Within this continuum, services and data are strategically positioned to optimize the end-user experience, particularly in scenarios marked by high user mobility and escalating demands for video quality. Consequently, video streaming applications running in such infrastructures can effectively avoid network congestion and other challenges, ensuring superior quality of experience for end-users.

This chapter, therefore, is dedicated to providing an overview of the aforementioned technology and concepts, focusing on the edge-cloud continuum infrastructure as the bedrock for supporting video streaming applications and tackling its associated challenges. We also provide an introduction and overview of the content steering architecture defined in the Dynamic Adaptive Streaming over HTTP (DASH) [1] protocol as a core mechanism for steering incoming video requests to appropriate services placed closer to end users. Finally, we present the technologies that are often used at the platform level to manage the platform's underlying devices and support service placement.

Furthermore, this chapter presents a GitHub [1] repository with a fully functioning video streaming application that leverages the content steering architecture to better exploit resources in the edge-cloud continuum. We provide all the necessary code for the interested reader to perform experiments with video streaming and content steering architecture in their machines.

[1] `https://github.com/robertovrf/content-steering-tutorial`

The remainder of this chapter is organized as follows: Sec. 1.3 presents a definition of the Edge-cloud Continuum and discusses the challenges and opportunities of such distributed infrastructures; Sec. 1.3 details the video streaming protocols and introduces the content steering architecture; Sec. 1.4 introduces the main technologies and tools used to manage clusters on the Edge-cloud continuum; Sec. 1.5 presents the available repository with a fully functioning video streaming application; Sec. 1.6 discusses the challenges and research opportunities for supporting video streaming applications on the edge-cloud continuum; and finally, Sec. 1.7 concludes the chapter.

1.2. Exploiting the Edge-cloud Continuum Resources

In this section, we introduce the concept of edge-cloud computing continuum and contextualize how new technologies are needed to exploit the resources in the computing continuum enabled by such infrastructures. To conclude, we define how these infrastructures support video streaming applications and the challenges involved at the platform and service level in exploiting the underlying resources available at the infrastructure level.

1.2.1. Defining the Edge-cloud Continuum

The Edge-cloud Continuum is a hierarchical and highly heterogeneous infrastructure with devices spread throughout a wide geographical location. It ranges from centralized cloud computing platforms located at the core of the network extending to end-user devices located at the edge of the network, resulting in a computing continuum.

Bittencourt et al. [10] and Peng et al. [46] have described the computing continuum stemming from the development of the fog/edge computing technology created to aggregate and process data generated by the Internet of Things (IoT) devices. A different and possible definition stems from the perspective of service placement from the cloud platform to devices executing closer to end users. Fig 1.1 illustrates the Edge-cloud Continuum concept, giving an overview of the infrastructure and its main characteristics.

The hierarchical edge-cloud infrastructure is represented in the aforementioned figure, where end-user devices are placed at the bottom of the hierarchy. These devices have an important characteristic that mainly characterizes the computing continuum. End-user devices often have constrained resources and small network latency [49]. Examples of these devices are sensors, household appliances (e.g., smart TVs), laptops and desktop computers, smartphones, autonomous vehicles, drones, and many other computing devices.

In the second layer of the infrastructure (LAYER 2..N), we have the edge computing platform [57]. These are clusters of computers located in close proximity to end-users. These clusters of computers have more available computing resources (e.g., CPU and storage) than the user devices but add a bit more network latency. They are also often used as the base for Content Delivery Networks (CDN) [70, 39, 12] due to their strategic location close to the end user and thus presenting small network latency while presenting reasonable and available computing resources. The infrastructure of servers presented as the second layer can extend further as a set of clusters that connects one geographic region to the cloud. This second layer can be extended to many layers up to the cloud.

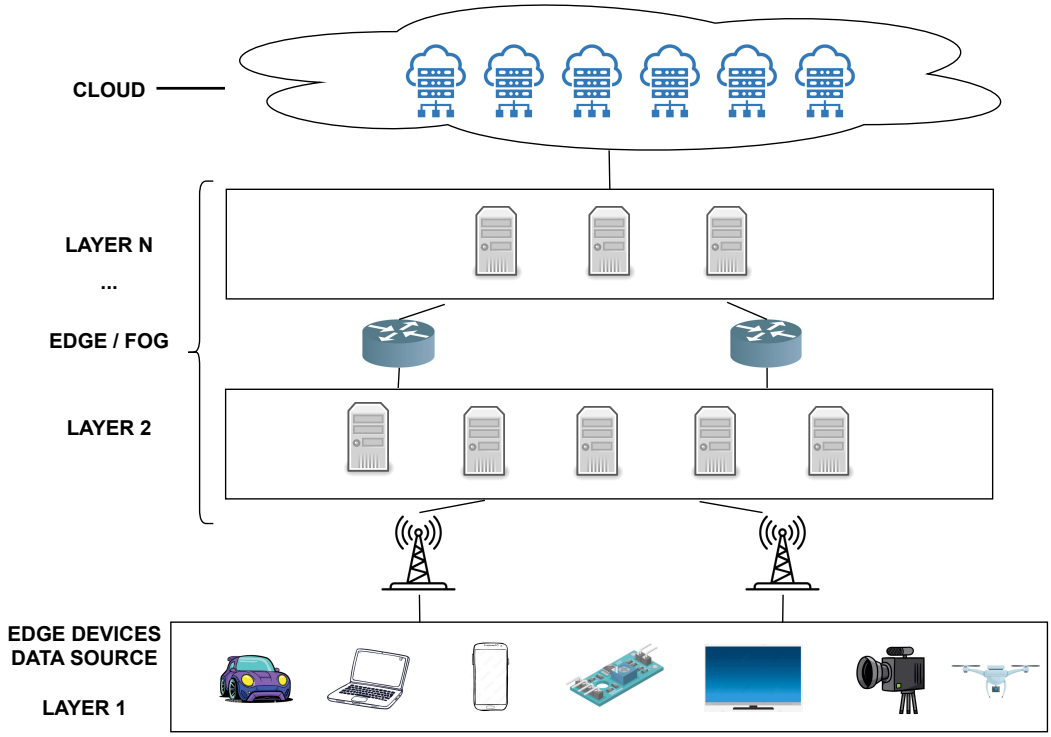


Figure 1.1. Overview of Edge-Cloud Continuum Infrastructure.

The cloud computing platform is located at the top layer. At this layer, the infrastructure has the characteristic of having the illusion of unlimited computing resources while maintaining a high network latency. At this level, CPU and storage-intensive services are better accommodated, but due to the distance between end users and the services, there is an added network latency in response time.

The resulting edge-cloud infrastructure has been explored to support services for systems that range from machine learning-based, virtual and augmented reality applications to video streaming, autonomous vehicles and others in recent years. The growing popularity of such infrastructures makes deploying and managing applications in such infrastructures crucial for supporting contemporary systems.

The development of systems to exploit resources in the edge-cloud continuum is marked with many challenges. These challenges mainly involve monitoring services throughout the continuum, supporting strategic service and data placement across edge and cloud to better exploit computing resources as the demands of the application and users change over time.

1.2.2. Exploiting the Hierarchy of Computing Resources

In order to explore the computing continuum infrastructure, systems face many challenges [10, 41]. One of the main challenges is to equip software deployed on the continuum to be able to move around the infrastructure to better exploit the available resources.

This is a challenging task that involves anticipating the resource demands of ap-

plications and the unexpected changes of those needs as changes occur in the operating environment. For instance, if the systems' workload volume suddenly spikes or the resources suddenly disconnect from the network, the executing software is required to adapt in order to properly exploit the underlying resources.

Moreover, besides adapting in the face of unexpected changes, software executing on the continuum should also adapt in anticipation of events. For instance, in a scenario where users are moving (i.e., user mobility) we often can apply forecasting strategies to predetermine the route where the users may take [2]. Especially if we consider a user's daily routine. In such cases, we can anticipate where to place services and data to better exploit resources in the continuum to increase users' Quality of Experience (QoE) as they move.

In general, building applications to exploit the resources in an edge-cloud continuum infrastructure involves many important challenges. We now present the main ones and discuss possible directions for future research. The main challenges we present in this section are the actual challenge of managing and properly allocating resources in the continuum, the challenge of abstracting the complexity of programming and deploying services capable of moving from cloud to edge to end-user devices and vice-versa, and finally, the challenges of monitoring services across the continuum.

1.2.2.1. Resource Allocation and Optimization

Exploiting computing resources in the continuum is crucial for optimizing systems performance. Although resource allocation is a classical distributed systems problem [30], it gains new dimensions when considering the scale of edge-cloud platforms.

The resource allocation problem usually requires schedulers that implement models to receive the set of available resources as input, and the application demands and outputs a plan for when and where to execute the application. In the edge-cloud continuum setting, resource allocation becomes not only difficult due to the number of constraints and requirements from applications and the sheer number of heterogeneous resource capabilities but also due to the volatility of its operating environment.

In the edge-cloud continuum, it is not sufficient to have schedulers implementing fixed decision-making models, but instead to have models that can adapt and evolve to consider changes in the operating environment. Theses changes includes the addition and removal of computing resources, end user mobility, incoming workload pattern changes, and so on.

Furthermore, to tackle the scalability in terms of the number of devices present in such deployment environments, we divide resource allocation tasks among controllers. This allows schedulers to act on a specific set of devices, considering requirements and constraints from a specific set of services, reducing the complexity of the model. This leads to a problem of where to place such controllers and how they should cooperate to optimize the global system's goal instead of optimizing isolated parts of the system.

Relevant strategies have been discussed in the literature to tackle the volatility issue when making resource allocation decisions in volatile environments [61, 15]. More-

over, different coordinating strategies have been explored to make joint decisions in multiple controllers over distributed infrastructures [19, 11, 45].

1.2.2.2. Service and Data Placement

Placing services and data on the different devices that compose the edge-cloud continuum is essential to maintaining a high Quality of Service (QoS) for certain systems. Some applications require services that are commonly executed on the cloud to be moved and executed at the edge closer to users or move services from end-user devices to the cloud when the service requires more resources.

Thus, strategically placing services and data in specific parts of the continuum is an important task to optimize applications running on the continuum. Regarding service mobility, many supporting technologies have gained popularity. Container technologies and container orchestration technologies provide a platform to enable general adaptation, including mobility. For instance, services can be wrapped inside Docker [2] containers and deployed in any cluster on the continuum managed by Kubernetes [3].

Furthermore, it is possible to see a trend in the industry to create stateless services to facilitate the exploitation of containers and container orchestration as a supporting platform for adaptation. A stateless service is a component that retains no information on previous requests or users' sessions and thus can be moved and replicated throughout the infrastructure with no changes to its implementation. Due to being stateless and self-contained, microservice-based architectures [31, 25, 26] and Function-as-a-Service [56, 58] have gained popularity in contemporary software systems and are widely used in the edge-cloud continuum.

However, not every service can be developed with no state. Many services require access to state information to compose a functioning system. In this context, engineers carefully design services that carry no state but are highly dependent on data stored in a cloud database. In these settings, the service is free to exploit the underlying platform to be freely replicated without maintaining state consistency across its replicas, but it suffers from performance degradation when placed far from the database on which it depends.

These data-dependent services can still be created as microservices or using a programming model such as Function-as-a-Service. They can also still leverage the container-orchestration platform underneath it to be placed in different devices across the distributed infrastructure. In a setting where the edge and cloud seamlessly integrate, we observe a significant decline in performance when these services are deployed on an edge node to facilitate rapid access from user devices. This is because the service keeps constantly accessing (updating or fetching) data in the cloud, adding network latency to transfer data to/from the cloud, defeating the purpose of placing the service closer to end users.

As a response to this problem, recent research has focused on the implementation of the concept now known as stateful Function-as-a-service (FaaS) [48]. Using this, the

[2]Docker: Container Technology - `https://docker.com`

[3]Kuberbetes: Container Orchestrator - `https://kubernetes.io`

functions can move to the edge, and access replicated data that is now copied to the edge. Likewise, similar strategies also support data access on the edge to microservice-based solutions [50].

1.2.2.3. Monitoring in the Edge-Cloud Continuum

In order to make precise service placement decisions and allocate resources on the continuum, a key challenge is to collect the correct metric based on which placement decisions will be based. In an ever-changing environment such as the edge-cloud continuum, an important task is monitoring the correct metric at the correct location and reporting the collected information to the decision-making component in time.

The edge-cloud continuum operating environment demands a highly adaptive and flexible solution for monitoring metrics. This is due to the constant changes that affect different aspects of the environment. Changes in the workload pattern and volume dictated by the way users interact with the system, changes in the devices on the infrastructure – new devices are integrated into the platform or functioning devices become unavailable, changes regarding the mobility of users, and so on.

These changes require the system to accommodate the new operating environment conditions to maintain a high-quality user experience. As the system changes, the components responsible for monitoring the system must change accordingly. For instance, if services are replicated or placed in different parts of the infrastructure, the monitoring system should adapt accordingly to collect data to support further decision-making. Therefore, edge-cloud continuum infrastructures require monitoring solutions that can adapt to the location where they collect metrics, the type of metric they collect, and how they process and make the collected information available to the decision-making component.

Adaptive monitoring solutions have been the focus of many research over the years [22, 63, 71, 4]. These research papers explore the requirements for creating adaptive monitoring systems, the mechanisms to support such adaptation, and the dimensions of the monitoring that need adaptation according to the scenario and environment in which the system is deployed. However, few initiatives tackle the end-to-end engineering challenges for deploying and managing adaptive monitoring solutions at the scale of the edge-cloud continuum, the most relevant presented by Colombo et al. [16].

Despite the few initiatives exploring an end-to-end solution, many technologies have been developed for monitoring services and data in the edge-cloud continuum. The review conducted by Verginadis [67] presents an interesting overview of many of such technologies describing Prometheus [4], Datadog [5], Zabbix [6] and Elastic Stack [7].

In the end, besides the scale on which these monitoring solutions are required to operate when deployed over edge-cloud continuum infrastructures, these tools should also consider the sheer levels of heterogeneity for monitoring the highly diverse devices

[4]

[5] `https://www.datadoghq`.

[6] `https://www.zabbix.com`.

[7] `https://www.elastic.co/elastic-stack`

that compose the continuum. That entails the application of different techniques to probe different metrics from different devices.

1.2.3. Computing Resource Demands for Video Streaming Applications

In the remainder of this chapter, we will focus on exploring service placement across the edge-cloud continuum to support video streaming applications. As the foundation of the explored approach, we leverage the Content Steering [1] architecture to exploit resources on an edge-cloud continuum infrastructure for video streaming.

Particularly for video streaming applications, the edge-cloud continuum infrastructure offers a great range of computing resources to accommodate the ever-increasing demands of end users [28]. Considering the data volume, Internet connection quality, and mobility of end users, systems are required to adapt to new operating conditions.

Current architectures to support video streaming solutions consider the placement of video content in CDNs. This data placement closer to where the end users are connected reduces the network latency for the users to access the desired content. This helps increase the quality of experience for the end user. Many video streaming solutions use multi-CDNs hosting copies of video content and have user players directly fetch video content from specific CDNs closer to the end user using specific URLs.

HSL [8] and DASH-IF [9] [59] have proposed and implemented a content steering architecture to determine which CDN the video player is fetching data during video playing. This gives the player more flexibility in adapting where the video is being fetched. Particularly in scenarios with user mobility or when the number of users increases, content steering can assist in balancing the request load from one location to another, reducing network congestion while increasing the quality of experience for end users.

Content-steering architectures are very useful when considering the placement of video streaming services across the continuum. Over the next sections, we discuss how to leverage such architecture to steer video content requests to specific services placed throughout the continuum. This, in turn, opens the possibility for a more flexible and adaptive infrastructure capable of handling video streaming to millions of users.

Finally, edge-cloud continuum infrastructures are also ideal for scaling content-steering solutions, as we can place different steering controllers across the infrastructure. This strategic placement of content steering controllers can act on a specific network region, reducing the number of users handled per controller and allowing this solution to scale drastically to accommodate millions of users worldwide.

1.3. Video Streaming Architecture Overview

This section investigates the mechanisms and structures that support video data transmission within network infrastructures. Section 1.3.1 presents a vision from encoding video content into digital formats to optimizing data packets for continuous delivery. Each component within these architectures ensures a seamless viewing experience for end users. In Section 1.3.2, we designed an architecture for Content Steering Services. Initially, a cen-

[8]HTTP Live Streaming (RFC 8216)

[9]Dynamic Adaptive Streaming over HTTP Industry Forum

tralized steering server orchestrates operations, followed by an Edge-Cloud Continuum approach. The service's main objective is to show users' directness to a video streaming service in real-time, balancing the load and improving scalability.

1.3.1. Adaptive Video Streaming

Although traditional video delivery methods have their merits, they face certain limitations when it comes to connection-oriented protocols like Real-Time Messaging Protocol (RTMP/TCP) [65] or Real-time Transport Protocol (RTP/UDP) [54] for connectionless protocols. Typically, the media server pushes the media to the client in these protocols. However, these methods require help scaling the infrastructure, depending on specific vendors, and having high maintenance costs necessitating complex and expensive servers. Over time, new technologies have emerged to overcome these limitations and enhance video streaming [8]. One of the most popular solutions is HTTP Adaptive Streaming (HAS), which offers several benefits over traditional methods. Adaptive streaming is a dynamic video delivery technique that adjusts the video playback quality in real time based on the user's network conditions, with the primary goal of providing seamless viewing playback by adapting to changes in available bandwidth.

Dynamic Adaptive Streaming over HTTP (DASH) [60] and Apple's HTTP Live Streaming (HLS) [44] are two of the most popular HAS solutions in use today. These methods, which handle over half of all video streams, are particularly prevalent in on-demand and live streaming platforms. Figure 1.2 depicts the basic workflow concept of a HAS service for video streaming involves the video source transmitting the raw video data to the HAS servers, which then encode it to produce multiple representations of a single video.

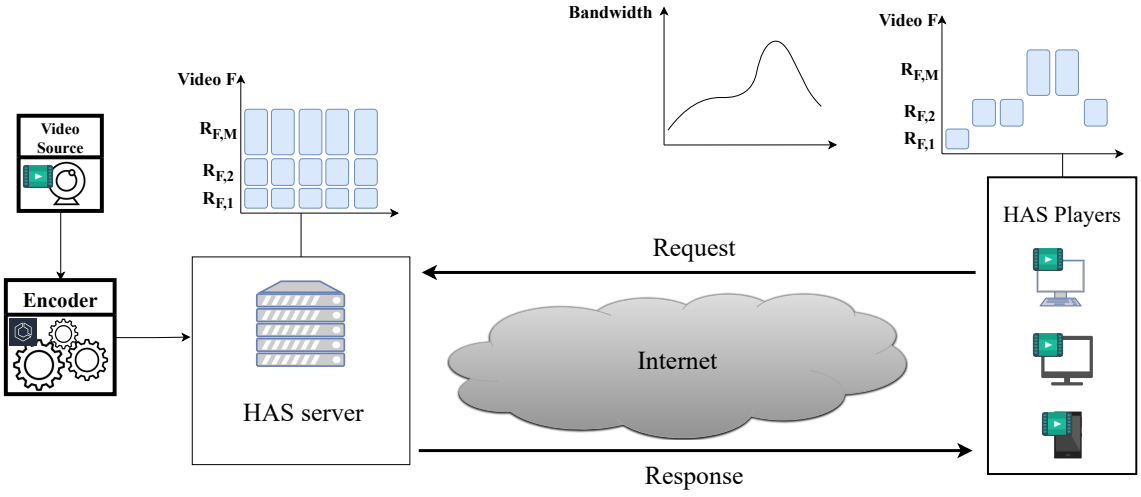


Figure 1.2. Overview of typical HTTP adaptive streaming server. The server provides the media segments in different representations. The client requests media segments in desired representations and measurement of throughput and buffer fill level.

In the context of HAS, videos are divided into short segments, typically lasting one to ten seconds. The encoder then encodes each segment into different versions with varying resolutions or bitrates. The Media Presentation Description (MPD) manifest file

keeps track of the storage location of these segments on a server. When a user watches a video, their HAS player utilizes this information, internet speed, and device capabilities to select the best version to stream. This process of segmentation and encoding is a key aspect of HAS, enabling it to adapt to the user's network conditions and provide a seamless viewing experience.

Different coding formats can encode the video, such as Advanced Video Coding (AVC), High Efficiency Video Coding (HEVC), Versatile Video Coding (VVC), VP9, and AOMedia Video 1 (AV1). These codecs generate multiple encoded representations of the video at different bitrates, catering to diverse network bandwidths. Clients requesting the video content can seamlessly download segments from these encoded representations. The client-side player intelligently selects the appropriate bitrate based on real time network conditions and dynamically switches between representations to ensure smooth playback. The downloaded segments are then concatenated within the playback buffer and processed by a standard decoder for video rendering. This process ensures a conforming bitstream compatible with decoders on various client devices.

AVC, commonly known as H.264, is a foundational video compression standard developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). Its widespread adoption owes to its versatility and compatibility across various platforms. However, despite its widespread use, AVC must improve to achieve optimal compression efficiency for high-resolution and high-quality video content.

HEVC, also known as H.265, emerges as the successor to AVC, aiming to address its predecessor's shortcomings and meet the increasing demand for higher quality video at lower bitrates. Developed by the same organizations responsible for AVC, HEVC introduces advanced encoding techniques such as larger block sizes, improved motion compensation, and more efficient entropy coding. Consequently, HEVC offers substantially improved compression efficiency, making it an attractive choice for various applications, including ultra-high-definition television (UHDTV), video streaming, and mobile video. Despite its advantages, HEVC adoption has been hindered by licensing issues, particularly in terms of royalties.

VP9 and AV1 are open and royalty-free video compression formats. They offer comparable compression efficiency to HEVC while remaining accessible and free. As a result, open-source implementations find widespread use in web browsers, online video platforms, and streaming services. However, some devices and platforms may limit adoption due to the need for more hardware support.

VVC, known as H.266, emerges as the latest generation video compression standard. VVC represents a significant leap in compression efficiency compared to previous standards, offering enhanced support for high-resolution video, HDR content, and immersive multimedia experiences. While VVC demonstrates remarkable compression gains, its adoption may face challenges due to licensing issues and the need for hardware support. Despite these challenges, VVC holds promise for applications requiring the highest compression efficiency and quality in video delivery.

1.3.1.1. Application-Network Interaction in Video Streaming

Regarding the HAS paradigm, video streaming applications are traditionally built upon the Transmission Control Protocol (TCP) [21]. Within this paradigm, application-layer Adaptive Bitrate (ABR) logic operates independently from the transport layer. While ABR logic dynamically adjusts streaming quality based on available bandwidth, TCP congestion control regulates data flow to avoid network congestion. However, the lack of coordination between these mechanisms often results in inefficiencies and concerns about fairness. ABR logic, which operates independently at the application layer, has a tendency to adjust the streaming quality too frequently without taking into account the network conditions monitored by TCP congestion control. This can lead to unfair bandwidth allocation among different sessions and can also cause instability in network utilization.

Additionally, uncoordinated adjustments by both mechanisms can exacerbate congestion issues instead of alleviating them [47]. To tackle these difficulties, an integrated approach is necessary wherein ABR logic and TCP congestion control cooperate to optimize resource allocation and improve network stability. By synchronizing these control loops and promoting communication between layers, it becomes possible to mitigate inefficiencies and ensure a fair bandwidth distribution among streaming sessions, thereby enhancing an overall users' QoE [42].

The emergence of the Quick UDP Internet Connections (QUIC) transport protocol, as presented in [34], has opened up new avenues for improving the interaction between video streaming applications and transport functionalities. While certain studies advocate for QUIC as a promising solution for video streaming, others engage in debates comparing it to TCP [55]. Furthermore, concerted efforts have been made to enhance QUIC's support tailored to video streaming applications.

A recent study [13] has evaluated QUIC's performance in the context of real time video streaming and found instances where QUIC exhibited excessive reliability, leading to suboptimal performance compared to TCP. QUIC performs better than the TCP protocol, where low-throughput use cases are necessary. However, in some cases, there are degradations compared to TCP and previous generations of HTTP, where the network bandwidth is poor or there is a variation in the bandwidth, as is typically experienced during mobility. The dynamics of application-network interaction for video streaming, particularly concerning integrating emerging transport protocols like QUIC, represent fertile ground for future investigation. Understanding these interactions and developing integrated solutions tailored to the specific needs of video streaming applications present exciting avenues for future research and innovation in this domain.

1.3.1.2. Bitrate Adaptive Schemes

Within the domain of client-server multimedia delivery, Bitrate Adaptation Schemes can be classified into client, server(s), network, and mixed-based adaptation. However, in this work, we limit our scope to client-based adaptation, representing most approaches [8]. These schemes delegate the client player to autonomously select the optimal video segment representation based on the network conditions. This selection process hinges on

one or more metrics, with available bandwidth and playback buffer health being the most common. Client-based adaptation schemes aim to achieve a seamless viewing experience by dynamically adjusting the bitrate of the streamed video. They strive to balance maintaining high video quality with mitigating the effects of fluctuating bandwidth, thereby preventing buffering stalls and ensuring smooth playback.

Within the domain of client-server multimedia delivery, Bitrate Adaptation Schemes can be classified into client, server(s), network, and mixed-based adaptation. However, this work limits our scope to client-based adaptation, representing most approaches [8]. These schemes delegate the client player to autonomously select the optimal video segment representation based on the network conditions. This selection process hinges on one or more metrics, with available bandwidth and playback buffer health being the most common. Client-based adaptation schemes aim to achieve a seamless viewing experience by dynamically adjusting the bitrate of the streamed video. They strive to balance maintaining high video quality with mitigating the effects of fluctuating bandwidth, thereby preventing buffering stalls and ensuring smooth playback.

In general, available bandwidth- and buffer-based adaptation suffers from poor QoE due to unreliable bandwidth estimation methods and long-term bandwidth fluctuations. Meanwhile, in mixed-based adaptation, the client selects bitrate using bandwidth, buffer occupancy, segment size, and duration metrics. Thus, many studies can develop complex algorithms to select the following segments. Among the studies in mixed-based adaptation, we highlight some studies that have adopted predictive strategies based on the past and current states of the player context as well as the device context to determine the next segment's bitrate [5, 38]. Meanwhile, in Bentaleb *et al.* [6], a Game Theory Adaptive bitrate scheme was developed. This technique employs a cooperative game in coalition form, enabling the clients to reach a consensus on the ABR selection process by formulating it as a bargaining process. Consequently, this approach enhances the viewer's QoE and ensures video stability.

Another movement noted in academia is the increasing popularity of Learning-based schemes, which benefit from the latest advancements in machine learning techniques [32, 69, 7]. Learning-based schemes can derive effective strategies without making any assumptions about the environment. However, their performance heavily depends on the quality of the training data. Since network environments can be diverse, and their dynamics change over time, predicting future states accurately requires more effort. Additionally, implementing learning-based schemes on constrained devices is impractical due to their high storage and computational costs.

1.3.2. Content Steering Server-based Design Architecture

Designing a video streaming architecture requires a distinct approach, especially when considering the involvement of multiple content delivery networks (CDNs), each operating independently with its multimedia contents. CDNs currently play a vital role in delivering media content to end users. They can replicate their content across numerous locations or deploy distributed servers to retrieve content from a central source. Such features imply that designing a video streaming architecture entails thorough planning and coordination.

However, CDNs have limitations when it comes to the network edge. Some CDNs may only provide coverage in some relevant regions, while others may face internal capacity constraints. Moreover, some CDNs may need more caches to support the effective delivery of extensive video collections to the target audience. As a result, some works are utilizing dynamic "CDN switching" technologies to adjust content delivery paths for streaming dynamically.

To address the growing demand for streaming services, the technical communities responsible for developing prominent streaming standards have devised a system where every end user regularly contacts a central manager, the content steering server. End users receive instructions on which CDN to retrieve the content during each interaction. The steering server implements a load balancing strategy based on user feedback and collected global metrics. This strategy can balance long-term business logic and short-term QoE maximization.

In order to explain how the Content Steering protocol mechanism operates, we provide the manifest files for both HLS and DASH locations for CDNs and the Steering Server during a streaming session. We will use the manifest files in 1.1 and 1.2. In these examples, we will use `https://cdn.com` as the CDN server and `https://steeringserver.com` as the content steering server. The examples here demonstrate only a single CDN server, although multiple tags can implement it.

```
1 #EXTM3U
2 #EXT-X-CONTENT-STEERING:SERVER-URI="https://steeringserver.com"
3 #EXT-X-STREAM-INF:BANDWIDTH=1280000,PATHWAY-ID="cdn"
4 https://cdn.com/hi/video.m3u8
5 #EXT-X-STREAM-INF:BANDWIDTH=1280000,PATHWAY-ID="cdn-2"
6 https://cdn-2.com/hi/video.m3u8
```

Listing 1.1. HLS manifest

```
2 <BaseURL serviceLocation="cdn">https://cdn.com/</BaseURL>
3 <BaseURL serviceLocation="cdn-2">https://cdn-2.com/</BaseURL>
4 <ContentSteering defaultServiceLocation="cdn" queryBeforeStart="true">
5 https://steeringserver.com
6 </ContentSteering>
```

Listing 1.2. HLS manifest

The manifest file received by the user also contains metadata from steering elements. HLS/DASH streaming players seamlessly detect the presence of steering servers and initiate communication with them throughout the streaming session. They send HTTP GET requests to the specified steering server indicated in the manifest, potentially incorporating additional parameters, such as throughput and the pathway utilized by the user.

In this example, the response contains two servers identified by the PATHWAY-PRIORITY array. These servers are capable of providing the requested video segments. It prioritizes the EdgeCache server, while the CDN server has a lower priority. The specifications of the EdgeCache are detailed in the PATHWAY-CLONES with an original CDN base, and the edge node address rules are given by the URI-REPLACEMENT, with the host replaced by a URL in HOST.

```
{
"VERSION": 1,
"TTL": 10,
"RELOAD-URI": "https://steeringserver.com?session=abc",
"PATHWAY-PRIORITY": ["EdgeCache", "CDN", "CDN-2],
"PATHWAY-CLONES": [
   {
       "BASE-ID": "CDN",
       "ID": "EdgeCache",
       "URI-REPLACEMENT": {
           "HOST": "https://edgecacheserver.com",
       }
   },
    {
        "BASE-ID": "CDN",
        "ID": "EdgeCache",
        "URI-REPLACEMENT": {
           "HOST": "https://edgecacheserver.com",
        }
    }
]
}
```

Listing 1.3. Json response.

The client receives these instructions with a Time-To-Live (TTL) of 10 seconds, indicating the response interval for requesting the next update. Reducing response time is crucial for enabling many additional system utilities. Thus, when TTL becomes shorter than the size of the player's buffer, this automatically enables QoE optimizations, such as buffering prevention or allowing clients to use higher-quality streams.

This callback mechanism ensures timely adjustments to the streaming pathways based on network conditions or nodes' availability. Shorter response times are critical for different network congestion adaptations, such as fault-tolerant, mobility, and many other applications.

This syntax for steering server responses and client-server interactions remains consistent for both HLS and DASH systems. Consequently, a single server can manage content steering operations for both protocols. For detailed specifications regarding the response format within the Steering response, refer to the guidelines outlined in [1].

Within this multimedia service inside the Edge-Cloud Continuum, the architecture of video streaming systems emerges as a dynamic ecosystem comprising different components. These components include client-side elements, CDNs, edge cache servers, origin servers, and steering servers. Together, they form a network of agents working seamlessly to ensure efficient content delivery. Each stage in this multimedia delivery step significantly influences end users' QoE. The concept of Content Steering on the Edge-Cloud Continuum addresses the dynamism of edge computing environments and enables applications to exploit edge-cloud computing resources better. In this Section, our survey focuses on the key aspects of the distribution process, firstly, in a centralized steering server in the cloud. Then, we integrated the service with the Edge-Cloud Continuum.

1.3.2.1. Centralized Steering Server-based Design

In a Centralized Steering Server-based design, a single entity executes the load balancing strategy in content delivery. This design's workflow typically involves achieving some beneficial effect. For example, it may perform fault tolerance control through the video streaming servers, increasing the system's reliability. It may also perform CDN load balancing, enabling broader distribution.

We can note some limitations. The first one is scalability. As the number of clients and concurrent requests increases, the centralized steering server may encounter scalability challenges. Managing a large volume of client requests and dynamically directing them to the optimal CDN in real time requires substantial computational resources and network bandwidth. Scaling up the steering server infrastructure to accommodate growing demand can take time and effort. Let us assume, for example, that we have an event watched by 1M of concurrent viewers. Then, with 10 seconds TTL, the steering server must process at least 100K requests per second. That is a high number, conventional hardware, and some non-trivial logic required for deriving each steering response; it may easily overload a single server or a cluster of servers.

Maintaining and operating a centralized steering server infrastructure incurs significant ongoing costs. These costs include hardware acquisition, network infrastructure expenses, software development, and maintenance. Such expenses can be considerable, as operating costs can escalate rapidly as the system scales to accommodate more clients and higher traffic volumes.

A prolonged TTL diminishes the usefulness and efficiency of content steering mechanisms. Although a long TTL may be suitable for basic load balancing and CDN management, more is needed for other critical objectives such as QoS and QoE optimizations. It may also compromise the speed of fault tolerance responses. For example, when clients experience buffering issues during content delivery, redirecting them to an alternative CDN after a long delay may not effectively mitigate the buffering problem.

1.3.2.2. Content Steering at the Edge-Cloud Continuum

Leveraging edge and cloud resources allows for greater scalability and flexibility in content delivery infrastructure. Edge servers can handle localized spikes in demand and deliver content closer to end-users, while cloud resources provide additional capacity and support for global content distribution. Overall, Content Steering at the Edge-Cloud Continuum offers speed, scalability, reliability, and personalization, enabling organizations to deliver content more efficiently and effectively to end users worldwide.

Figure 1.3 illustrates the proposed Edge-Cloud Steering implementation. The design consists of two stages. In the first stage, the cloud steering master has its load balancing strategy given the list of CDN servers. They also decide which regional edge steering server should take responsibility for the user in the next TTL turn. Once a request arrives, the master node sends the response based on the list of CDN and edge steering server URLs. In the second stage, the system makes subsequent steering decisions at TTL intervals for each streaming session. Edge computing platforms implement all oper-

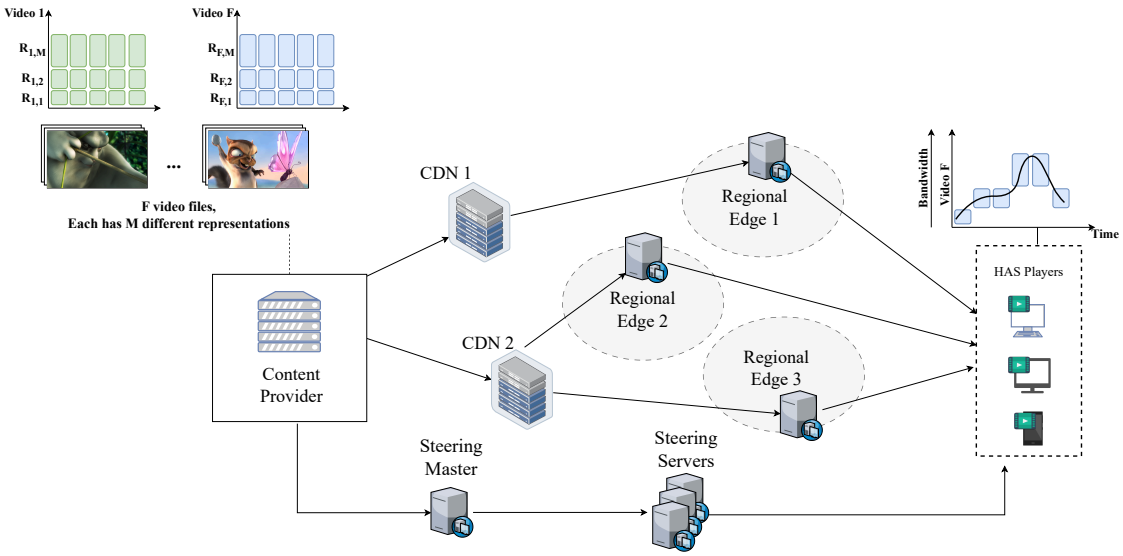


Figure 1.3. Overview of Content Steering at the Edge-Cloud Continuum: Directing user requests to edge-based content delivery systems, leveraging real-time network conditions and user proximity.

ations. This approach shows great potential in addressing the limitations of a centralized approach [24, 20]:

- **Scalability:** This implementation achieves scalability levels comparable to CDNs or platforms responsible for executing edge functions. By distributing processing tasks across multiple edge points, it can seamlessly accommodate growing demands without compromising performance;
- **Cost-Efficient Deployment:** Deploying this implementation becomes significantly more economical due to reduced bandwidth and per-request costs at CDNs or edge platforms compared to the higher egress traffic costs associated with cloud platforms. This cost-effectiveness makes it a more viable option for organizations seeking efficient content delivery solutions;
- **Enhanced Responsiveness:** The two-phase implementation enhances responsiveness by enabling lower TTL between clients and servers. This reduced latency facilitates quicker interactions between users and content servers, leading to smoother content delivery experiences.

Reducing response times is paramount for unlocking additional functionalities within the system. When TTL values become shorter than the size of the player's buffer (e.g., 1-30 seconds), it automatically enables QoS and QoE optimizations. For instance, it helps prevent buffering issues and allows clients to access higher quality streams seamlessly. Moreover, shorter response times are essential for supporting mobile applications, disaster recovery scenarios, and various other use cases requiring swift and efficient content delivery.

1.4. Managing Clusters on the Edge-cloud Continuum

In IoT and Edge Computing environments, resources are limited and heterogeneous. Choosing technologies that provide the best experience in relation to the orchestration and execution of the proposed services is one of the main points of attention in this scenario. This section will cover the technologies used to accomplish this in the context of this short course, highlighting the container approach, its orchestration through Kubernetes and how both were used to implement the Edge Computing infrastructure used.

1.4.1. Introducing Containers

Containerization is a lightweight virtualization technology that enables the encapsulation of services and applications in small standard units. Unlike traditional machine virtualization, where a virtual machine is instantiated with the entire operating system, libraries and other dependencies, the container encompasses only the application (or service) of interest and the dependencies necessary for its execution. Scalability is enhanced using this approach, as containers consume less resources and generate lower overheads when compared to virtual machines. Each container is defined from an image, built on a stacked layer structure, where each change in the subsequent layer creates a new layer. Except for the topmost layer, all the others are read-only and can be reused to compose other images. Containers share the same kernel and isolate application processes from the rest of the operating system.

In this context, Docker[10], CoreOS and other interested parties in the container industry established the Open Container Initiative (OCI) with the purpose of defining standard specifications for container technologies. Currently, OCI proposes 3 standards: runtime, image and container distribution. Therefore, OCI allows compatibility among all artifacts produced in compliance with the proposed standards. Among the use cases related to comply with OCI, we highlight Internet of Things and Edge Computing, which are closely related to this short course.

The container technology stands out for offering distinguished characteristics in relation to other virtualization approaches, namely: portability, volatility, and resource consumption. Portability is evident in the way containers are built, as the application is encapsulated with all the dependencies necessary for its execution, a container will have the same execution behavior in any environment in which it is provisioned. Regarding volatility, due to the ephemeral nature of the container, all possible data persistence is carried outside the container. Therefore, in case of execution issues related to a failed container, it is not necessary to carry out debugging to identify the fundamental problem affecting its functioning, but rather to destroy it and create a new instance without the need to take care of application data, which can be costly. Finally, as previously described, as it is an environment with a small number of additional processes and services, as opposed to virtual machines, the container consumes fewer resources, optimizing the physical resources available on the host system.

Fig.1.4 shows the main differences between containers in relation to virtualization based on virtual machines.

[10]Docker is a container platform: `https://www.docker.com/`

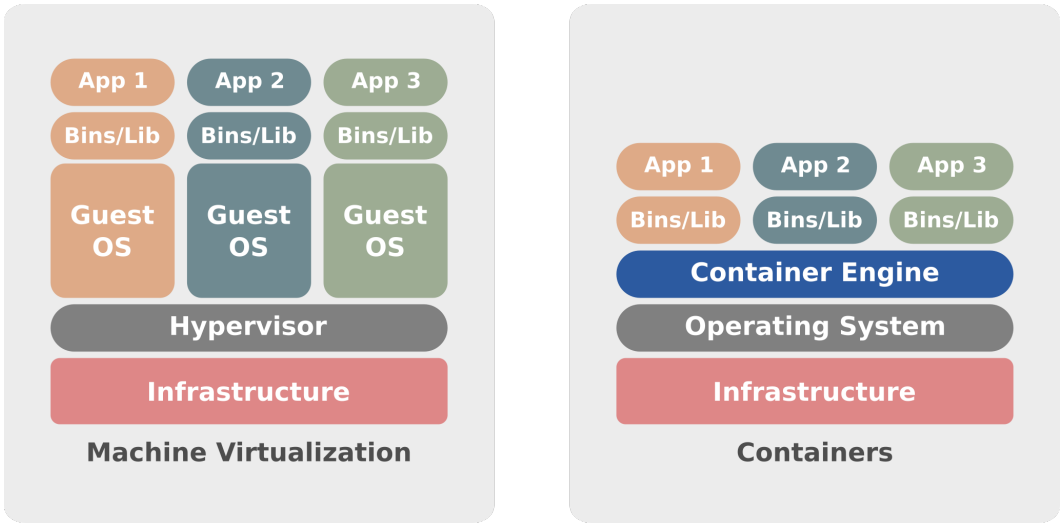


Figure 1.4. Containers vs Virtualization based on Virtual Machines.

It is important to note that running containers also require a runtime, the most famous of which is Docker. Container runtimes can be classified into two types: low-level and high-level. Low-level runtimes adhere to OCI specifications. Among them are runC, crun, runhcs, and containerd. On the other hand, there are runtimes that have an extra layer, because they present more functionalities than the minimum enough for just running containers. Examples of these high-level runtimes are Docker, Podman and CRI-O. For a runtime to execute an image of interest, a container image should be passed to the runtime. In the context of containers, images are stored in repositories that are referenced to the runtime at execution time. These repositories, also known as Registries, are well-structured and can be public or private. In the first case, the stored images are open to the public. In the second case, the images are made available to a specific audience, for example, one company employee; therefore, most of the time, they are available only through access credentials. Container image repository implementations include Red Hat's Harbor and Quay.

Containers enable the creation of a new application development paradigm based on microservices. This paradigm enables splitting application components into a number of relatively small services that, unlike paradigms with monolithic approaches, address the architecture and organization of applications by dividing the whole implementation into small, well-defined, and concise units according to the functions they perform in the application. The microservices communication is performed through well-defined APIs.

1.4.2. Managing Containers Lifecycle with Container Orchestrators

Containers are virtualized environments that offer several possibilities for use. Even though they offer a set of positive characteristics, they also bring challenges in managing their entire lifecycle in a simple way. In addition, containers alone do not deliver all the properties expected in the micro-services paradigm, being just one component that supports the paradigm. Properties such as elasticity and resilience are not delivered by container runtimes alone. To fill these gaps, orchestrators emerged, such as Docker Swarm, for example, to support containers lifecycle management (creation, replicas, garbage collector, removal of containers, etc.), among the orchestrators. A container orchestrator that has established itself as a market standard due to its versatility and effectiveness is Kubernetes[11].

The Kubernetes container orchestrator is an open-source project hosted by the Cloud Native Computing Foundation (CNCF). It is a well-documented orchestrator with an active community of developers and a growing set of funding entities (IBM, Google, Oracle, among others). There are currently several Kubernetes distributions, in which companies insert a set of functionalities into Vanilla Kubernetes (community version hosted at CNCF), and deliver them to interested parties through usage subscriptions. The main available Kubernetes distributions are Red Hat, Rancher, and VMware Tanzu.

When it comes to deployment architectures, Kubernetes is a highly distributed and flexible orchestrator. Composed of *Nodes*, it allows the orchestrator deployment to have specialized Nodes, making the organization as complex as needed; on the other hand, the same Kubernetes can be instantiated in an all-in approach – with just one Node assuming all responsibilities for orchestrating the environment. Among the types of Nodes provided by Kubernetes, two are considered fundamental and are present in most deployment architectures, which are the Worker Nodes and Control Nodes. Worker Nodes contain the computing resources that will be consumed by user workloads, while Control Nodes have the components related to the control of the entire environment and the orchestration services themselves (scheduling, for example). An interesting feature of Kubernetes is that even if the Control Nodes become unavailable, the workloads running on the Worker Nodes will continue to be active (even though with some restrictions). As it is organized into Nodes, the insertion of new Nodes, both workers (more resources) and Control, is a trivial task.

In the context of the workloads, unlike simple container runtime environments, in Kubernetes the smallest processing unit is a POD. A POD comprises one or more containers and a set of metadata that defines parameters related to the containers themselves (labels, the quantity of replicas, the quantity of resources required for execution, etc.), as well as the orchestrator's behavior over the POD. Considering the microservices paradigm, the POD corresponds to a microservice. A notable feature of PODs is that all containers in a POD share the same address space, so they can communicate through the localhost. Furthermore, this behavior allows the creation of some design patterns: the sidecar. This pattern allows the user to provide an auxiliary container that complements or adapts certain functionalities of the main container (the microservice). For example, given a microservice that logs the events of its execution to *syslog*, but the log aggregation

[11] Kubernetes is a container-orchestrator: `https://kubernetes.io/`

software used does not understand this protocol, in this scenario it would be possible to implement a sidecar to receive the syslog protocol on localhost, translating the protocol and sending it to the observability environment.

Kubernetes approaches the maintenance of the desirable state and the theory of controls as the core of its activities on PODs. Through these two approaches, Kubernetes, in control cycles (reconciliation cycle), checks whether the PODs running in the environment are in the desirable state. The respective desired states of each POD are defined through a structure called replica set. This object allows the indication of how many replicas of the POD of interest should exist at a given time. For example, if a replica set of a POD *A* is indicated as 2, two instances of POD *A* will be running. If a problem occurs and one of the replicas becomes unavailable, in the next reconciliation cycle a new POD *A* is automatically instantiated. In addition to the replica set, there are other objects that are defined around the POD and allow controlling the behavior regarding the deployment and execution of PODs, namely Deployment, StatefulSet, and DaemonSet. All of them use the Replica Set in their operation. Deployment operates mainly at the replica layer, and it can indicate the volumes used, number of POD replicas, among other properties. In addition to the properties and resources defined in Deployment, StatefulSet addresses a shortcoming in container environments, namely ephemerality. In StatefulSet, Kubernetes persists both network and volume address (resource location). Therefore, if a StatefulSet POD becomes unavailable, when Kubernetes instantiates it again (reconciliation cycle), it returns to using the network and volumes used by the POD that became unavailable. Finally, the Daemonset also offers the properties and resources defined in Deployment. Added to this, this object guarantees that each Kubernetes Node will receive at least one instance of the POD of interest. Through this approach, a POD can interact directly with the Kubernetes Node environment. A classic use case for Daemonset is to serve as a deployment model for log event collectors.

Kubernetes is intended to deliver complex tasks through trivial definitions. In this brief description of the main features it is clear that the use of Kubernetes is currently essential in the context of the use of containers.

1.4.3. Setting up a Kubernetes-based Cluster for Edge-Cloud Infrastructures

The edge-cloud computing continuum is a structure of heterogeneous devices (large capacity servers, access points, repeaters, among others) that can be arranged hierarchically in layers. The hierarchical distribution can include n-tiers, with a spectrum that goes from the upmost level (where the public or private cloud is located) to the bottommost layer. Considering a bottom-up approach, the bottommost layer contains the devices that end users interact with (smart TVs, smartphones, tablets, laptops, etc.). The intermediary tiers are composed of a collection of servers placed close to end-user devices, running back-end services close to data sources. Finally, the topmost layer are the cloud platforms, where all the resource-intensive back-end services run, exploiting the cloud platform's illusion of infinite computing resources available (e.g. storage, CPU, network , etc.).

A characteristic of the hierarchical structures of the edge-cloud computing continuum is that each layer devices are subject to their respective network latency and computational capacity when communicating with other devices. Lower layer devices, as they

are closer to end users, are subject to lower latencies when communicating with end users. Conversely, devices in the highest layer are subject to larger latencies resulting from networking devices processing and queueing as well as link medium propagation delays, as they are further away from the end user. Considering resource capacity, devices closer to end users often have limited computing resources, and as we move upwards through the layers, we have devices with more capacity available. Therefore, properly managing the balance between network latency and resource availability is crucial to supporting applications that demand a high-quality end-user experience.

The balance between network latency and computing resources is often achieved through service allocation and offloading strategies. As already mentioned, as these are heterogeneous layers considering the latency and computational capacity of the devices, to support the allocation of services in the different layers, lightweight and flexible execution environments are often used. In this scenario, containers and Kubernetes play a fundamental role in this type of environment.

1.4.3.1. Reference Infrastructure Model

To demonstrate the flexibility of using containers and the Kubernetes orchestrator in the edge-cloud computing continuum scenario, a reference infrastructure model was created consisting of three layers structured in a binary tree topology. The root of the tree is in the first layer, being composed of a node representing the cloud. In the second, intermediate layer, two nodes are present. And finally, in the lower layer, four nodes are present (Fig. 1.5). In this last layer, network latency is minimal when accessed by users. This topology ensures that nodes can only interact with their child nodes or parent nodes, not having access to other nodes in the cluster. This arrangement makes it possible to adequately simulate the hierarchy formed in real edge-cloud computing continuum infrastructures. It is possible to use virtualized environments to emulate devices in a real environment. In this reference model, infrastructure resources were deployed on a cloud server managed by OpenStack.

Kubernetes is the container orchestrator of choice for the reference model. Each of the tier nodes is managed by a Kubernetes instance. These instances are composed of an all-in model, with the Master and Worker operating in a single unit. Therefore, Kubernetes is responsible for container deployment and container lifecycle management – services deployed in this environment run inside Docker containers. Additionally, tools such as Prometheus are used to monitor consumed and available resources and node service analysis. Surveys collected by Prometheus are made available to application-level services to decide how to adapt application services to better accommodate workload volume, infrastructure resource allocation, and end-user mobility.

The reference infrastructure model reflects the structure presented in the topology diagram presented in the above-mentioned figure. There are seven nodes organized hierarchically in a tree. Some layer 2 nodes interfaces are used only when installing the environment (for example, to download packages and container images). We do not need external access to these nodes during the execution of the experiments. So, the networks interfaces from this nodes (1 and 2 - layer 2) must be removed after installing the environ-

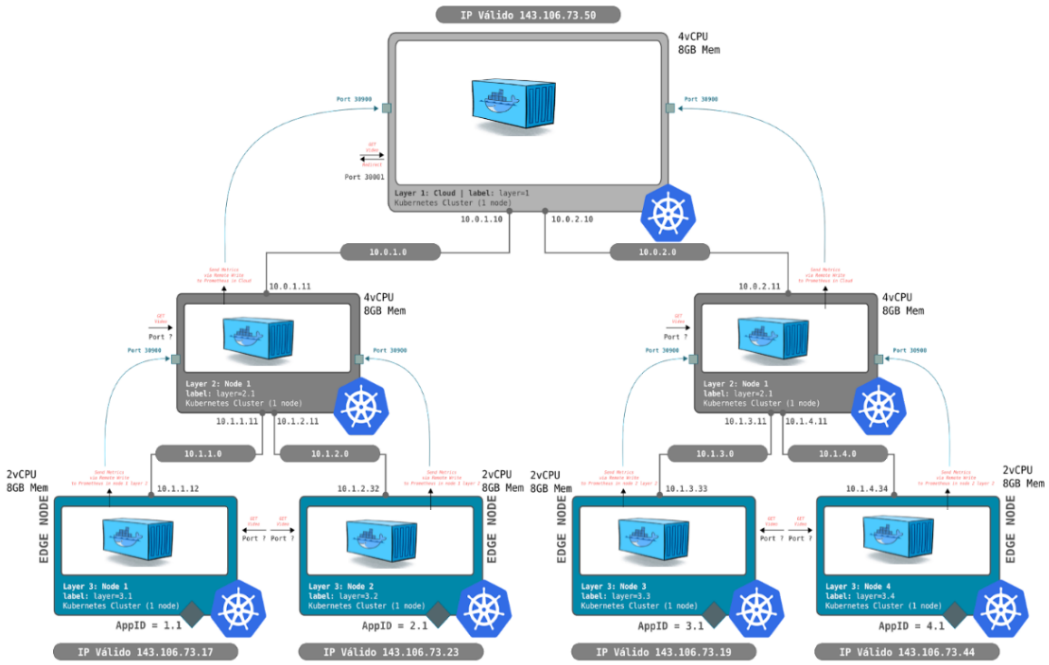


Figure 1.5. Topology diagram.

ment. In this model, two deployment and installation tools were mainly used: Terraform and Kubespray. Terraform was used to prepare virtual machines and networks in the OpenStack environment. Kubespray was used to install a Kubernetes cluster on previously created virtual machines. It forms a total of 7 virtual machines, each of which runs an all-in Kubernetes cluster instance.

Terraform is an open-source tool offered by HashiCorp. This tool allows the implementation of infrastructure environments described as code, i.e. Infrastructure as Code (IaC). Terraform encapsulates all the logic and APIs of the environment of interest and describes it as a provider. Such providers are used to make API calls abstractly to the implemented code. This makes the maintenance, support and migration process to other providers easier with Terraform. In the test scenario, the OpenStack provider and Terraform were used to configure and instantiate all the necessary resources.

To start the Terraform environment, the following steps should be followed where ten files will be created:

```
$ cd terraform/

## Download files related to the OpenStack provider:
$ terraform init

## Create the action plan. All changes to the OpenStack
## environment will be displayed here:
```

```
$ terraform plan

## Execute the previously created action plan:
$ terraform apply
```

- **00-variable.tf**: file with the definitions of the variables that will be used by the other resources. In this file, the value of FLOATING_IP_POOL must be changed with the floating pool of IPs that the infrastructure is using:

```
variable "floating_ip_pool" {
  type    = string
  default = "FLOATING_IP_POOL"
}
```

- **10-ssh-key.tf**: defines the public key (rsa type) that will be used to access the created virtual machines instances. In the entry below, the value of PUBLIC_KEY must be set to a public key.

```
resource "openstack_compute_keypair_v2" "user_key" {
  name       = "access-key"
  public_key = "PUBLIC_KEY"
}
```

- **20-network.tf**: file with definitions about the network aspects used in the rest of the environment.
- **30-security-groups.tf**: security groups used. Initially, all ports are closed. In OpenStack (provider used), opening is carried out through Security Groups.
- **40-flavor.tf**: definition of the set of resources used in virtual machines (flavor). This definition is made in terms of vCPU, memory and disk.
- **50-image.tf**: operating system image used in the environment's virtual machines. Some images presented in OpenStack providers have a limitation regarding network interfaces (1 in this case). This behavior can lead to network availability issues when more than one interface is configured. Neutron (Openstack's network service) tries to configure the presented interfaces, indicated by Terraform, but only the first one is defined successfully. It is recommended to use more modern OS images that allow the configuration of multiple network interfaces (e.g. Almalinux 8, CentOS, etc.).
- **60-instances.tf**: definitions of VM instances and security groups. Floating IPs (previously allocated in the Openstack infrastructure) are also configured. Regarding floating IPs, if they are not present and need to be allocated on demand, it is necessary to uncomment the resource below (inserted in the file).

```
#resource "openstack_networking_floatingip_v2" "fip_pool" {
#   count  = 7
#   pool   = var.floating_ip_pool
#   description = "Floating IP to project Ericson"
#}
```

As previously described, 7 nodes will be provisioned. Therefore, 7 floating IPs are needed for the initial configuration of the environment. Consequently, 7 entries like the one shown below must be present for the respective nodes.

```
resource "openstack_compute_floatingip_associate_v2"
"fip_associate_node11" {
   floating_ip = "177.220.85.224"
   instance_id = openstack_compute_instance_v2.vm_node_11.id
   fixed_ip    = "192.168.200.210"
}
```

- **provider.tf**: defines the provider that will be used (OpenStack). This provider's credentials can be defined in the file itself or in the cloud.yaml file.
- **clouds.yaml**: Settings and access credentials for the Openstack provider, used to provide the reference model infrastructure. This file can be obtained directly from the Openstack graphical interface (Horizon Dashboard).
- **versions.tf**: Defines the Terraform versions that will be used.

Kubespray: is an open source tool intended for deploying and managing Kubernetes clusters. The tool works with public cloud, on-premises, bare metal, and staging solutions, making it ideal for managing highly available clusters across multiple different platforms. It is comprised of Ansible playbooks, inventory, provisioning tools, and domain knowledge for generic OS/Kubernetes cluster configuration management tasks. Kubespray in the context of the reference model is used to provision 7 clusters. For each cluster there is an inventory with its respective configuration.

- **Node Cloud**: cluster-root/
- **Node 1 layer 2**: cluster-node2-1/
- **Node 2 layer 2**: cluster-node2-2/
- **Node 1 layer 3**: cluster-node3-1/
- **Node 2 layer 3**: cluster-node3-2/
- **Node 3 layer 3**: cluster-node3-3/
- **Node 4 layer 3**: cluster-node3-4/

After installing kubespray and its dependencies, each folders from the list above must be copied to *invetory/*. Preparing the environment for running kubespray is as follows:

```
## Clone the repository:
$ git clone https://github.com/kubernetes-sigs/kubespray.git
$ cd kubespray

```

```
## Copy the kubernetes cluster inventory files to the inventory
folder. REPOSITORY is set with this git project folder:
$ cp -r ${REPOSITORY}/artefacts/kubespray/cluster-* ./invetory/

## Create virtual environment:
$ python3 -m venv venv

## Activate the environment:
$ source venv/bin/activate

## Install the dependencies:
$ pip install -r requirements.txt
```

It is necessary to configure the network addresses as defined in Terraform. Below is an example configuration.

```
all:
  hosts:
    node1:
      ansible_host: 177.220.85.224
      ip: 192.168.200.210
      access_ip: 192.168.200.210
  children:
    kube_control_plane:
      hosts:
        node1:
    kube_node:
      hosts:
        node1:
    etcd:
      hosts:
        node1:
    k8s_cluster:
      children:
        kube_control_plane:
        kube_node:
    calico_rr:
      hosts: {}
```

Ansible_host must be defined with the floating IP address of the virtual machine that Kubespray (Ansible components) will access to perform the installation of the kubernetes node. The other two addresses (*ip* and *accessip*) must be of the internal interface, in this case the interface that is associated with the floating IP. Another necessary configuration is to define the docker repositories where the images will be obtained. This must be done in *inventory/CLUSTER_NODE/group_vars/all/containerd.yml*.

The following entries must be inserted at the end of the file:

```
(...)
containerd_registries:
  "docker.io":
    - "https://mirror.gcr.io"
    - "https://registry-1.docker.io"
\end{verbatim}

```

```
Once all configurations have been made, Kubespray must be run for each
    node of interest.
\begin{verbatim}
$ ansible-playbook -i inventory/cluster-root/hosts.yaml -u
almalinux -b -v --private-key="${KEYS_FOLDER}/id_rsa" cluster.yml

-i: inventory with de node cluster address, quantity of nodes,
kubernetes version and other configurations.
-u: user used by OS to execute the activities.
-b: become a root if necessary.
-v: verbose mode.
--private-key: file with private key to access the VM
```

After running Kubespray for each new node, it is necessary to verify that the environment was correctly installed. This must be performed for each of the Kubernetes clusters created. Within each virtual machine in the reference model, it is initially necessary to obtain the Kubernetes credentials, copy them to an appropriate location, and verify that the cluster is operational (1 node and the operational services PODs should be ready).

```
## Create the Kubernetes config file:
$ mkdir .kube
$ sudo cp /etc/kubernetes/admin.conf .kube/config
$ sudo chmod 666 .kube/config

## Check the kubernetes master node:
$ kubectl get nodes
NAME    STATUS   ROLES           AGE   VERSION
node1   Ready    control-plane   18h   v1.25.5

## Check the kubernetes system PODs:
$ kubectl get pods -n kube-system
NAME                                      READY   STATUS    REST-
calico-kube-controllers-75748cc9fd-ttdm8  1/1     Running   0
calico-node-z8cg9                         1/1     Running   0
coredns-588bb58b94-k2qrs                  1/1     Running   0
dns-autoscaler-d8bd87bcc-djvm8            1/1     Running   0
kube-apiserver-node1                      1/1     Running   1
kube-controller-manager-node1             1/1     Running   2
kube-proxy-t4b97                          1/1     Running   0
kube-scheduler-node1                      1/1     Running   2
nodelocaldns-9vk68                        1/1     Running   0
```

Some other settings may be needed, such as checking and setting host names and inserting them in */etc/hosts* since, there is no DNS server for the environment. It may also require some OS tuning. These details are beyond the scope of this document.

1.5. Adaptive Video Streaming Application for the Edge-cloud Continuum

To enhance the readers' understanding of the framework behind the content steering service distributed across the Edge-Cloud Continuum, we present a detailed case study focusing on a video streaming service hosted on a CDN platform within the Institute of Computing (IC) at UNICAMP. We aim to illustrate alternative video provisioning points for users. To achieve this, we examine a scenario where the CDN, hosted in the Cloud,

redirects users to an edge server on the local machine. Various factors could prompt this redirection, including recurrent congestion issues within users' home networks.

Fig. 1.6 illustrates the basic elements that a fully functioning video streaming application running on the edge-cloud continuum contemplates. We provide all necessary software with detailed instructions for the reader to execute the system on their machine. Moreover, the interested reader may use the available system to conduct their own experiments and analysis.

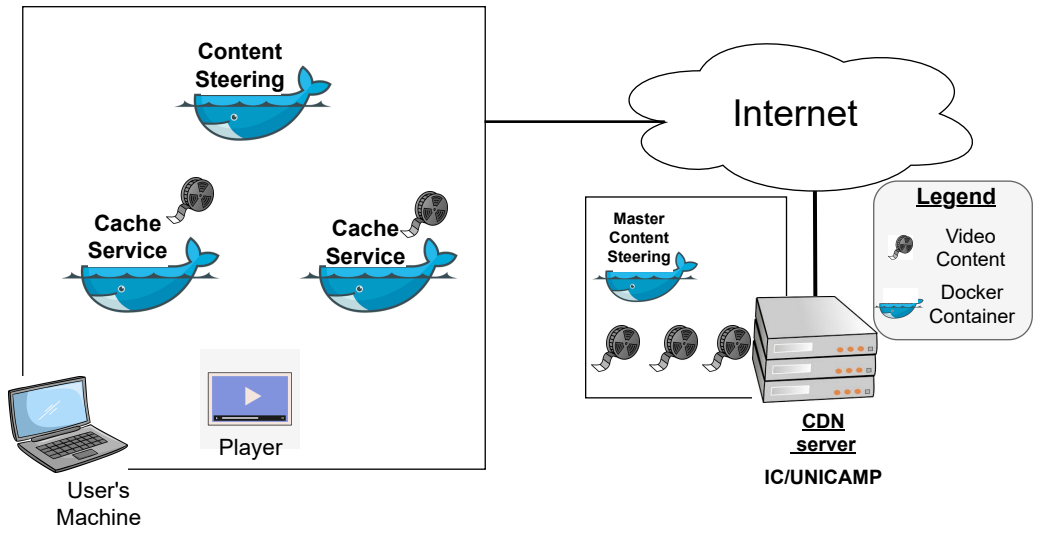


Figure 1.6. Main elements of the video streaming application. A local machine running a video player, two cache services and a content steering, connected to a CDN with video content and a master content steering.

The video streaming application we provide has four main basic components: i) the CDN servers where the main content and the master content steering are hosted, ii) the cache servers running on the reader's machine, as an illustration of cache servers deployed in multiple locations over the edge of the network, iii) the edge content steering module responsible for steering video content in a specific location of the edge-cloud continuum, and iv) the user's video player, that in this case also runs on the user's machine.

The CDN servers are placed in the IC-Unicamp dependencies, and their content should be accessible to the reader. Note that any CDN can be used as a replacement for the one we have set up to demonstrate our video streaming application. We provide a detailed description of how to set up and configure a CDN in case the reader wants to use their own settings. The CDN hosts both the video content and an executing master content steering service. The client's video player always contacts the master content steering and collects content from the CDN in the first round, later it connects to the local content steering and starts fetching content from the local cache service.

The cache services are provided in the form of Docker container images to facilitate the system's execution. The cache services in our scenario are different points for content provision. Thus, the video player can fetch video content directly from the cache services in the same network instead of fetching content from a distant CDN server.

We also make available a content steering service that should be executed on the reader's local machine. We provide both the Docker image and the source code. This content steering service is responsible for steering the user's video player to specific cache services in the reader's machine. In our video streaming systems example, this content steering represents the module responsible for adapting and handling content steering in a specific geographic location in a real-life deployment in a full-scale edge-cloud continuum infrastructure.

Moreover, the content steering module is also responsible for monitoring the cache service metrics and detecting when a specific cache service is down or not responding. When this occurs, the content steering redirects the video player to fetch information from another cache service or the closest CDN server. In our practical experiment, we can observe such behavior after deploying the system and stopping the Docker container running the cache service that provides the content to the video player.

Finally, we provide a fully functioning video player that implements the content steering protocol. The video player accesses the CDN server to fetch the manifest file with the information on where to fetch the video content. After the first interaction with the CDN node, the video player receives a URL for a local cache service and fetches the video content from the local cache service.

1.5.1. Repository Description

The complete materials, algorithms, tools, and tutorials for installing and running the application can be accessed on Github [12]. To obtain the code and materials, you must perform a process named clone, which downloads the source code. We recommend using Ubuntu version 22.04 since the setup instructions are tailored for it.

After the cloning process, a directory named content-steering-tutorial should have been created. From the repository's root directory, folders for the video streaming and steering services and a dataset folder containing the multimedia content should be created. To streamline the execution of experiments, we provide some shell scripts to automate the process.

1.5.2. Running the Streaming Services

Now we are initiating video streaming and steering microservices. It's assumed that the reader is familiar with Docker. To start the video streaming containers, we create two replicas of the service using the command:

```
1 $ docker compose -f steering-service/docker-compose.yml up -d
```

Listing 1.4. Command to create the streaming services.

If you wish to configure the number of replicas, modify the replica count within the docker-compose.yml file in the streaming-service directory. Now, ascertain the IPs assigned to each service and associate them with the domains created in /etc/hosts.

You can then verify the features of local video streaming. Remember that our focus is on Content Steering features. Therefore, our experiments use a straightforward

[12] https://github.com/robertovrf/content-steering-tutorial

```
$ docker compose -f steering-service/docker-compose.yml up -d
```

Listing 1.6. Command to create the steering service.

HTTP server to serve static multimedia content. This approach allows us to observe the services functioning autonomously, independent of others. Then, you can access the *dash.js* player and try to load the manifest using the URL format:

```
https://<streaming-domain>/<streaming-path>/manifest.mpd
```

Listing 1.5. URL to watch the video in any Dash player.

Replace <*streaming-domain*> with your streaming domain and <*streaming-path*> with the path to your video.

1.5.3. Running the Steering Service

To create the Content Steering service, we follow similar commands to those used for the streaming service setup. Within the steering-service folder, you find the *docker-compose.yml* file containing the default configurations for local steering creation.

The Content Steering mechanism involves real-time monitoring of deployed containers. Various tools for resource monitoring within containers have been proposed, including docker stats, Prometheus, and Google's cAdvisor. In this tutorial, we utilize the docker package to collect resource usage metrics from containers and determine the availability of video streaming services. The steering collects the Docker stats metrics from the containers with running stats every two seconds. It is recommended that the reader investigate the *steering-service/src/monitoring.py* code to understand metrics capture.

The communication to the steering service occurs via an HTTP GET method, as discussed in Section 1.3. We have set up a basic Flask application with a single route handling GET requests at the root URL, running on port 30500. Listing 1.7 provides the code snippet responsible for managing these requests (*steering-service/src/app.py*). When a user initiates a request to the content steering, the service identifies active edge servers using the *getNodes()* method from monitoring in line 7. Subsequently, the algorithm updates the list with the selected edge servers and the target server, presenting them as the output for the user's request. The parser method constructs the message (lines 9-14). The return incorporates essential components such as the TTL and the RELOAD-URI, which specify the URI for reloading the steering service's configuration as needed. Finally, we use the jsonify function to convert the message into JSON.

```
1  @app.route('/', methods=['GET'])
2  def do_get(name):
3
4  trg = request.args.get('_DASH_pathway', default='', type=str)
5  thr = request.args.get('_DASH_throughput', default=0.0, type=float)
6
7  nodes = _monitoring.getNodes()
8
9  message = _dash.parser(
10     target  = trg,
```

```
$ ./create_certs.sh <streaming-domain>
```

Listing 1.8. Command to generate certificates.

```
    nodes   = nodes,
    uri     = BASE_URI,
    request = request
)

return jsonify(message), 200
```

Listing 1.7. Main method from the steering service.

1.5.4. Running the Testbed

In the initial experiment, the localhost environment was set up using Docker tools. Start by modifying your local host's file, found at */etc/hosts*, to assign custom domain names for the streaming services you intend to create. After that, execute the script *create_certs.sh* to generate certificates and activate HTTPS for each streaming service in localhost:

Next, retrieve a DASH video from the dataset provided by [64] and store it in the specified folder named "dataset". The dataset offers a range of codec choices, including AV1, AVC, HEVC, and VVC. You may utilize wget or any other suitable method to download the video. We used the "Eldorado" video encoded in AVC with 4-second segments for our experiments.

1.5.5. Running the Video Player

We can test our experiment once we have deployed the necessary components mentioned earlier. Our next step would be to request a video from a scenario where the CDN, located in the Cloud, serves as the initial source. As the user watches the video, they can switch between servers autonomously. This enables us to evaluate our edge servers' efficiency and effectiveness in delivering high-quality content.

To gain a better understanding of how the transition from the cloud-based content source to the edge servers works, we can utilize the player available at [13]. This player lets us visualize how the user's playback experience changes as they switch between servers.

1.6. Challenges and Opportunities

As we can now create and configure a Kubernetes-based cluster over the edge-cloud continuum, deploy and perform (re)placement of video streaming services over the infrastructure to improve the quality of experience, we start facing new and interesting challenges.

In this section, we discuss three of the main challenges we consider the most important: content steering optimization, the concept of zero-touch networks and their related challenges, and the issues that arise when considering user mobility.

[13]Content Steering - `https://reference.dashif.org/dash.js/latest/samples/advanced/content-steering.html`

1.6.1. Content-Steering Optimization

The content steering problem is an interesting and challenging problem that needs to be addressed. Especially in the edge-cloud continuum settings, where content steering has to handle high volumes of video request content. Thus, the content steering optimization problem can be formulated as follows:

Problem: *Given a set of users requesting video content and a set of CDN servers, how to best steer user requests to load balance them among the available servers considering user and server location, network congestion, video quality, and network bandwidth to maximize user quality of experience.*

A version of this problem has been tackled in the literature. Gama et al. [29] has presented the challenge and provided an Integer Linear Programming (ILP)-based solution. In the context of video streaming over edge-cloud continuum infrastructures, however, it is crucial to consider the volatile and dynamic nature of edge-cloud continuum infrastructures, where changes constantly and unexpectedly occur at the infrastructure, platform, and service levels and based on user behavior and request patterns. In such scenarios, the content steering policy needs to be constantly updated.

Therefore, static optimization solutions for content steering that are defined pre-deployment are not sufficient. The adoption of models to drive content steering is required to evolve over time to include changes in content popularity, user behavior, varying request volumes, and sudden network congestion, to name a few sources of unexpected events.

The application of machine learning, specifically reinforcement learning strategies, is desired in such volatile settings. This is because reinforcement learning techniques define their models as the agent interacts with the environment; in this case, the content steering orchestrator defines which CDN server the users will fetch the content from. This is crucial in situations where the content steering orchestrator encounters unexpected scenarios where it is required to optimally load balance the incoming requests to the available CDN servers, for instance, a CDN server becomes unavailable or there is a sudden spike in requests in a specific geographic location.

An interesting line of research is to enable autonomous learning for both content steering policy and content steering orchestrator placement. This would enable the definition of where in the edge-cloud continuum to place a content steering orchestrator to content steering for a set of users while, at the same time, enabling the recently placed orchestrator to learn the best policy to steer incoming requests.

Autonomous learning solutions, such as the one outlined here, could help tame the increasing complexity of exploiting the underlying edge-cloud continuum resources while supporting the executing services for video streaming applications.

1.6.2. Zero-touch Network and Service Management (ZSM)

The application of closed-loop automation to bootstrap, configure, and optimize networking systems led to the definition of the concept of Zero-touch Networks and Service Man-

agement (ZSM) [40, 62]. Many research papers that address the concept of the zero-touch network come from the 5G/6G communities [9, 62, 18], given the rising complexity in setting up, configuring, and optimizing such networks.

As an important line of research, we have the application of closed-loop automation to set up, configure, and optimize video streaming services, platforms, and the underlying networking infrastructure to support video streaming applications in edge-cloud continuum settings. Besides the application of machine learning techniques to solve the content steering problem (Sec. 1.6.1), video streaming could benefit from autonomous (self-adaptive, self-optimizing, self-healing) techniques. They can be used to autonomously set up, perform placement, and optimize cache services throughout the edge-cloud continuum, even autonomously choosing the cache replacement strategy that best suits the cache service based on the handled requests.

Furthermore, the application of such techniques can autonomously define parameters in the content steering architecture. For example, adjust the parameter that defines how often players provide information to content steering orchestrators to make decisions on which server the player should fetch information. These parameters are currently set manually, and it has an impact on how fast the system adapts to avoid networking congestion as well as on the amount of information content steering orchestrator receives in a timespan.

A complementary line of research is to apply Intent-based Networks (IBN) [37] and Intent-driven Networks (IDN) [43, 23] concepts to guide zero-touch network optimization and service management. As we apply autonomous mechanisms to optimize the network, the use of high-level network intents defined by both application developers and network administrators can be used as network goals for the closed-loop automation mechanism to satisfy with minimum to no human intervention.

Examples of applying Intent-based Network techniques are described in many different networking contexts, such as IoT network management [17], 5G cloud service provision [3], and video service assurance [52]. Therefore, exploring similar techniques in the edge-cloud continuum to set up and optimize video streaming applications presents a fruitful direction for future research.

1.6.3. Exploring Adaptation to Accommodate User Mobility

An important challenge to tackle in this domain is to address user mobility. As users move from location *A* to location *B*, many changes are required to take place throughout the infrastructure, platform, and service level to maintain the quality of streaming.

User mobility has been a concern for video streaming for a while now, with papers proposing solutions to cope with mobility dating back to 3G networks [36]. The problem with handling mobility has not been fully addressed but rather intensified over the years. Despite the fact that networking infrastructures are becoming faster with 5G deployments and the expectations of 6G networks, the volume of streaming content, their unprecedented quality, and the number of users continue to increase, aggravating the problems related to video streaming services in the context of user mobility.

Solutions considering caching content on the edge [14] and/or attempting to pre-

dict user mobility [68] have been explored in the literature [35]. However, there are still opportunities to explore these approaches at the scale on which edge-cloud continuum infrastructures operate while streaming high-quality videos with an incredibly high number of mobile users consuming video content.

An interesting line of research is the exploration of machine learning approaches to identify and predict user mobility patterns and execute video prefetching to cache services deployed close to their location. Moreover, the combination of prefetching solutions based on user mobility patterns in tandem with data placement considering content popularity amongst groups of users moving towards the same direction may increase the quality of streaming and optimize computing resource allocation.

Finally, the exploration of prefetching solutions and handling unexpected user behavior is also an interesting research direction, especially considering the volatile nature of the edge-cloud continuum infrastructure. In the case of user mobility, some users may detour from their expected predefined route and change mobility patterns from time to time. In these cases, reactive approaches to deal with unexpected situations are desired and expected to work in tandem with proactive approaches, where prefetching/preventive adaptive strategies are in place.

1.7. Conclusion

This chapter presented an overview of the technologies and concepts involved in supporting video streaming over edge-cloud continuum infrastructures. In particular, we have introduced the concept of edge-cloud continuum and discussed the main challenges and opportunities. The discussions included the infrastructures provided, the main video streaming protocols, the concept of the content steering architecture, and the technology used to deploy and configure platform-level management tools to control the underlying computing resources. It was also presented a hands-on practical experiment with detailed instructions to execute a fully functioning video streaming application using content steering.

We hope to have contributed to the community by supplying a starting point for researchers to explore the diversity of challenges presented in deploying and managing video streaming applications over the Edge-cloud Continuum. We also hope that the technical material we supply on our GitHub [14] repository will facilitate the creation of testbeds for further experiments with content steering architecture and video streaming applications on the edge-cloud platforms.

Acknowledgements

This work was partially supported by the São Paulo Research Foundation (FAPESP), grant 2021/00199-8, CPE SMARTNESS, and by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

References

[1] Content steering for dash, 2022. Accessed 28-oct-2023.

[14] GitHub: `https://github.com/robertovrf/content-steering-tutorial`

[2] A. T. Akabane, R. Immich, R. W. Pazzi, E. R. M. Madeira, and L. A. Villas. Exploiting vehicular social networks and dynamic clustering to enhance urban mobility management. *Sensors*, 19(16):3558, Aug 2019.

[3] F. Aklamanu, S. Randriamasy, E. Renault, I. Latif, and A. Hebbar. Intent-based real-time 5g cloud service provisioning. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, 2018.

[4] F. Alessi, A. Tundo, M. Mobilio, O. Riganelli, and L. Mariani. Reprobes: An architecture for reconfigurable and adaptive probes, 2024.

[5] B. Alt, T. Ballard, R. Steinmetz, H. Koeppl, and A. Rizk. Cba: Contextual quality adaptation for adaptive bitrate video streaming. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1000–1008, 2019.

[6] A. Bentaleb, A. C. Begen, S. Harous, and R. Zimmermann. Want to play dash? a game theoretic approach for adaptive streaming over http. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, pages 13–26, New York, NY, USA, 2018. Association for Computing Machinery.

[7] A. Bentaleb, M. Lim, M. N. Akcay, A. C. Begen, and R. Zimmermann. Bitrate adaptation and guidance with meta reinforcement learning. *IEEE Transactions on Mobile Computing*, (01):1–14, mar 5555.

[8] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann. A survey on bitrate adaptation schemes for streaming media over http. *IEEE Communications Surveys & Tutorials*, 21(1):562–585, 2019.

[9] C. Benzaid and T. Taleb. Ai-driven zero touch network and service management in 5g and beyond: Challenges and research directions. *IEEE Network*, 34(2):186–194, 2020.

[10] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana. The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things*, 3-4:134–155, 2018.

[11] L. F. Bittencourt, A. Goldman, E. R. Madeira, N. L. da Fonseca, and R. Sakellariou. Scheduling in distributed systems: A cloud computing perspective. *Computer Science Review*, 30:31–54, 2018.

[12] R. Buyya, M. Pathan, and A. Vakali. *Content delivery networks*, volume 9. Springer Science & Business Media, 2008.

[13] S. Chaudhary, N. K. Shukla, P. Sachdeva, S. Chakraborty, and M. Maity. Managing connections by quic-tcp racing: A first look of streaming media performance over popular http/3 browsers. *IEEE Transactions on Network and Service Management*, pages 1–1, 2024.

[14] Y. Chen, H. Yu, B. Hu, Z. Duan, and G. Xue. An edge caching strategy based on user speed and content popularity for mobile video streaming. *Electronics*, 10(18), 2021.

[15] I. Cohen, C. F. Chiasserini, P. Giaccone, and G. Scalosub. Dynamic service provisioning in the edge-cloud continuum with bounded resources. *IEEE/ACM Transactions on Networking*, 31(6):3096–3111, 2023.

[16] V. Colombo, A. Tundo, M. Ciavotta, and L. Mariani. Towards self-adaptive peer-to-peer monitoring for fog environments. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '22, pages 156–166, New York, NY, USA, 2022. Association for Computing Machinery.

[17] B. M. Cordeiro, R. Rodrigues Filho, I. G. Júnior, and F. M. Costa. Steer: An architecture to support self-adaptive iot networks for indoor monitoring applications. *Journal of Internet Services and Applications*, 14(1):107–123, 2023.

[18] E. Coronado, R. Behravesh, T. Subramanya, A. FernÃ ndez-FernÃ ndez, M. S. Siddiqui, X. Costa-PÃ©rez, and R. Riggio. Zero touch management: A survey of network automation solutions for 5g and 6g networks. *IEEE Communications Surveys and Tutorials*, 24(4):2535–2578, 2022.

[19] A. Diaconescu, B. Porter, R. Rodrigues, and E. Pournaras. Hierarchical self-awareness and authority for scalable self-integrating systems. In *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 168–175. IEEE, 2018.

[20] J. Du, C. Jiang, A. Benslimane, S. Guo, and Y. Ren. Sdn-based resource allocation in edge and cloud computing systems: An evolutionary stackelberg differential game approach. *IEEE/ACM Transactions on Networking*, 30(4):1613–1628, 2022.

[21] W. Eddy. Transmission Control Protocol (TCP). RFC 9293, Aug. 2022.

[22] J. Ehlers, A. van Hoorn, J. Waller, and W. Hasselbring. Self-adaptive software system monitoring for performance anomaly localization. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 197–200, New York, NY, USA, 2011. Association for Computing Machinery.

[23] Y. Elkhatib, G. Coulson, and G. Tyson. Charting an intent driven network. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–5, 2017.

[24] R. Farahani, M. Shojafar, C. Timmerer, F. Tashtarian, M. Ghanbari, and H. Hellwagner. Ararat: A collaborative edge-assisted framework for http adaptive video streaming. *IEEE Transactions on Network and Service Management*, 20(1):625–643, 2023.

[25] K. Fu, W. Zhang, Q. Chen, D. Zeng, and M. Guo. Adaptive resource efficient microservice deployment in cloud-edge continuum. *IEEE Transactions on Parallel and Distributed Systems*, 33(8):1825–1840, 2022.

[26] K. Fu, W. Zhang, Q. Chen, D. Zeng, X. Peng, W. Zheng, and M. Guo. Qos-aware and resource efficient microservice deployment in cloud-edge continuum. In *2021*

IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 932–941, 2021.

[27] E. S. Gama, L. O. N. De Araujo, R. Immich, and L. F. Bittencourt. Video streaming analysis in multi-tier edge-cloud networks. In *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 19–25, Aug 2021.

[28] E. S. Gama, R. Immich, and L. F. Bittencourt. Towards a multi-tier fog/cloud architecture for video streaming. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 13–14, 2018.

[29] E. S. Gama, N. B. V, R. Immich, and L. F. Bittencourt. An orchestrator architecture for multi-tier edge/cloud video streaming services. In *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*, pages 190–196, 2023.

[30] A. Goscinski and M. Bearman. Resource management in large distributed systems. *ACM SIGOPS Operating Systems Review*, 24(4):7–25, 1990.

[31] Z. Houmani, D. Balouek-Thomert, E. Caron, and M. Parashar. Enabling microservices management for deep learning applications across the edge-cloud continuum. In *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 137–146, 2021.

[32] T. Huang and L. Sun. Deepmpc: A mixture abr approach via deep learning and mpc. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 1231–1235, 2020.

[33] R. Immich, L. Villas, L. Bittencourt, and E. Madeira. Multi-tier edge-to-cloud architecture for adaptive video delivery. In *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 23–30, Aug 2019.

[34] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[35] M. A. Khan, E. Baccour, Z. Chkirbene, A. Erbad, R. Hamila, M. Hamdi, and M. Gabbouj. A survey on mobile edge computing for video streaming: Opportunities and challenges. *IEEE Access*, 10:120514–120550, 2022.

[36] A. Kyriakidou, N. Karelos, and A. Delis. Video-streaming for fast moving users in 3g mobile networks. In *Proceedings of the 4th ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDE'05, pages 65–72, New York, NY, USA, 2005. Association for Computing Machinery.

[37] A. Leivadeas and M. Falkner. A survey on intent-based networking. *IEEE Communications Surveys and Tutorials*, 25(1):625–655, 2023.

[38] E. Liotou, T. Hoßfeld, C. Moldovan, F. Metzger, D. Tsolkas, and N. Passas. *The Value of Context-Awareness in Bandwidth-Challenging HTTP Adaptive Streaming Scenarios*, pages 128–150. Springer International Publishing, Cham, 2018.

[39] D. Liu, Z. Wang, and J. Zhang. Video stream distribution scheme based on edge computing network and user interest content model. *IEEE Access*, 8:30734–30744, 2020.

[40] M. Liyanage, Q.-V. Pham, K. Dev, S. Bhattacharya, P. K. R. Maddikunta, T. R. Gadekallu, and G. Yenduri. A survey on zero touch network and service management (zsm) for 5g and beyond networks. *Journal of Network and Computer Applications*, 203:103362, 2022.

[41] S. Moreschini, F. Pecorelli, X. Li, S. Naz, D. Hastbacka, and D. Taibi. Cloud continuum: The definition. *IEEE Access*, 10:131876–131886, 2022.

[42] V. Nathan, V. Sivaraman, R. Addanki, M. Khani, P. Goyal, and M. Alizadeh. End-to-end transport for video qoe fairness. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, pages 408–423, New York, NY, USA, 2019. Association for Computing Machinery.

[43] L. Pang, C. Yang, D. Chen, Y. Song, and M. Guizani. A survey on intent-driven networks. *IEEE Access*, 8:22862–22873, 2020.

[44] R. Pantos and W. May. HTTP Live Streaming. RFC 8216, Aug. 2017.

[45] M. L. M. Peixoto, T. A. L. Genez, and L. F. Bittencourt. Hierarchical scheduling mechanisms in multi-level fog computing. *IEEE Transactions on Services Computing*, 15(5):2824–2837, 2022.

[46] L. Peng, A. R. Dhaini, and P.-H. Ho. Toward integrated cloud-fog networks for efficient iot provisioning: Key challenges and solutions. *Future Generation Computer Systems*, 88:606–613, 2018.

[47] L. Peroni and S. Gorinsky. An end-to-end pipeline perspective on video streaming in best-effort networks: A survey and tutorial, 2024.

[48] T. Pfandzelter and D. Bermbach. Enoki: Stateful distributed faas from edge to cloud. In *Proceedings of the 2nd International Workshop on Middleware for the Edge*, MiddleWEdge '23, pages 19–24, New York, NY, USA, 2023. Association for Computing Machinery.

[49] F. Pisani, F. de Oliveira, E. S. Gama, R. Immich, L. F. Bittencourt, and E. Borin. Fog computing on constrained devices: Paving the way for the future iot. *Advances in Edge Computing: Massive Parallel Processing and Applications*, 35:22, 2020.

[50] C. Puliafito, C. Cicconetti, M. Conti, E. Mingozzi, and A. Passarella. Balancing local vs. remote state allocation for micro-services in the cloud-edge continuum. *Pervasive and Mobile Computing*, 93:101808, 2023.

[51] C. Quadros, E. Cerqueira, A. Neto, A. Riker, R. Immich, and M. Curado. A mobile qoe architecture for heterogeneous multimedia wireless networks. In *2012 IEEE Globecom Workshops*, pages 1057–1061, Dec 2012.

[52] C. E. Rothenberg, D. A. Lachos Perez, N. F. Saraiva de Sousa, R. V. Rosa, R. U. Mustafa, M. T. Islam, and P. H. Gomes. Intent-based control loop for dash video service assurance using ml-based edge qoe estimation. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pages 353–355, 2020.

[53] F. Santos, R. Immich, and E. R. Madeira. Multimedia services placement algorithm for cloud-fog hierarchical environments. *Computer Communications*, 191:78–91, 2022.

[54] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[55] M. Seufert, R. Schatz, N. Wehner, and P. Casas. Quicker or not? an empirical analysis of quic vs tcp for video streaming qoe provisioning. In *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 7–12, 2019.

[56] K. R. Sheshadri and J. Lakshmi. Qos aware faas for heterogeneous edge-cloud continuum. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pages 70–80, 2022.

[57] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[58] C. Sicari, D. Balouek, M. Parashar, and M. Villari. Event-driven faas workflows for enabling iot data processing at the cloud edge continuum. In *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*, UCC '23, New York, NY, USA, 2024. Association for Computing Machinery.

[59] D. Silhavy, W. Law, S. Pham, A. C. Begen, A. Giladi, and A. Balk. Dynamic cdn switching-dash-if content steering in dash. js. In *Proceedings of the 2nd Mile-High Video Conference*, pages 130–131, 2023.

[60] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.

[61] P. Soumplis, P. Kokkinos, A. Kretsis, P. Nicopolitidis, G. Papadimitriou, and E. Varvarigos. Resource allocation challenges in the cloud and edge continuum. In *Advances in Computing, Informatics, Networking and Cybersecurity: A Book Honoring Professor Mohammad S. Obaidat's Significant Scientific Contributions*, pages 443–464. Springer, 2022.

[62] N. F. S. d. Sousa and C. E. Rothenberg. Clara: Closed loop-based zero-touch network management framework. In *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 110–115, 2021.

[63] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovski. Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review. *Journal of Systems and Software*, 136:19–38, 2018.

[64] B. Taraghi, H. Amirpour, and C. Timmerer. Multi-codec ultra high definition 8k mpeg-dash dataset. In *Proceedings of the 13th ACM Multimedia Systems Conference*, MMSys '22, pages 216–220, New York, NY, USA, 2022. Association for Computing Machinery.

[65] M. C. Thornburgh. Adobe's RTMFP Profile for Flash Communication. RFC 7425, Dec. 2014.

[66] M. Uitto and A. Heikkinen. Evaluation of live video streaming performance for low latency use cases in 5g. In *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 431–436, 2021.

[67] Y. Verginadis. A review of monitoring probes for cloud computing continuum. In L. Barolli, editor, *Advanced Information Networking and Applications*, pages 631–643, Cham, 2023. Springer International Publishing.

[68] X. Wang, T. Kwon, Y. Choi, H. Wang, and J. Liu. Cloud-assisted adaptive video streaming and social-aware video prefetching for mobile users. *IEEE Wireless Communications*, 20(3):72–79, 2013.

[69] H. Yousef, J. L. Feuvre, and A. Storelli. Abr prediction using supervised learning algorithms. In *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6, 2020.

[70] Q. Yuan, H. Zhou, J. Li, Z. Liu, F. Yang, and X. S. Shen. Toward efficient content delivery for automated driving services: An edge computing solution. *IEEE Network*, 32(1):80–86, 2018.

[71] E. Zavala, X. Franch, and J. Marco. Adaptive monitoring: A systematic mapping. *Information and Software Technology*, 105:161–189, 2019.

Chapter 2

SmartNICs: The Next Leap in Networking

Marcelo Caggiani Luizelli (UNIPAMPA), Francisco Vogt (UNICAMP),
Guilherme Mendes Vieira de Matos (UFSCar), Weverton Cordeiro (UFRGS),
Alberto Egon Schaeffer Filho (UFRGS), Marcos Schwarz (RNP),
Fabio Luciano Verdi (UFSCar), Christian Esteve Rothenberg (UNICAMP)

Abstract

Programmable Network Interface Cards (SmartNICs) have gained increasing momentum due to their flexibility to offload complex networking tasks from the host CPU to a programmable hardware architecture. Despite the performance gains (e.g., lower latency), programming, debugging, and operating SmartNICs pose challenges due to diverse hardware architectures (e.g., System-on-Chip, FPGA, and ASIC), various programming languages, and operation modes. This tutorial aims to shed light on the design principles and operation of modern SmartNICs, covering hardware architectures, programming software ecosystem, performance capabilities and open research challenges in this evolving domain, The tutorial concludes with a hands-on experience involving cutting-edge SmartNICs such as Nvidia BlueField, Mellanox ConnectX, and Netronome NFP.

Resumo

As placas de rede programáveis (SmartNICs) têm recebido crescente visibilidade devido à sua flexibilidade para descarregar tarefas de rede complexas da CPU do host para uma arquitetura de hardware programável. Apesar dos ganhos de desempenho (por exemplo, menor latência), a programação, depuração e operação das SmartNICs apresentam desafios devido a diversas arquiteturas de hardware (por exemplo, System-on-Chip, FPGA e ASIC), várias linguagens de programação e modos de operação. Este tutorial tem como objetivo esclarecer os princípios de projeto e operação de aplicações em SmartNICs modernas, abrangendo arquiteturas de hardware, ecossistema de software de programação, capacidades de desempenho e desafios de pesquisa abertos neste domínio em evolução. O tutorial conclui com uma experiência prática envolvendo SmartNICs atuais, como Nvidia BlueField, Mellanox ConnectX e Netronome NFP.

2.1. Introduction and Motivation

The ability to program the network data plane has reshaped the network operations and management landscape, opening up a multitude of opportunities for delivering custom-made networking solutions [Kianpisheh and Taleb 2023]. By running home-brewed networking solutions within network programmable devices, network operators and practitioners have an opportunity to close the management control loop and instrument per-packet processing decisions, at line rate, in the order of nanoseconds.

Recently, there has been an increasing interest from both academy and industry towards programmable NICs (Network Interface Cards), commonly referred to as SmartNICs[1]. For instance, Nvidia (BlueFlied model), Netronome (Agilio model), Intel (IPU model), AMD (Pensando), Xilinx (Alveo models), and others are already battling for shares of this new market. These devices are gaining attention due to their ability to efficiently offload from the host CPU to the SmartNIC hardware a wide range of intricate networking tasks, as well as tasks unrelated to networking – promising high-performance packet processing while reducing the total cost of ownership. Examples of offload tasks include in-network caching [He et al. 2023] and in-network machine learning [Saquetti et al. 2021, Swamy et al. 2022] – just to name a few examples.

The term SmartNIC appeared first in the 1990s, when NICs were mostly used to connect computers to networks [Ponomarev and Ghose 1998]. They performed basic tasks like packet reception, transmission, and simple processing, such as checksum offloading. However, NIC offloading has since evolved to include more advanced and specialized capabilities that are necessary to meet the demands of modern networking environments. This is due to the increasing need for improved performance, efficiency, and scalability. For example, in the 2000s, NICs were developed with a TCP Offload Engine. Later, in the 2010s, NICs evolved to support virtualization requirements such as VLAN and RSS. Nowadays, NICs (or SmartNICs) support a long list of embedded functions, including advanced features such as RoCE (RDMA over Converged Ethernet), stateful connection tracking, or even storage acceleration [Xilinx 2024].

Despite the constant evolution of NICs, they only become "Smart" as networking vendors add some level of programmability to them. There are different levels of SmartNICs programmability. Some vendors allow rewriting the whole hardware description (e.g., FPGA-based SmartNICs), while others allow offloading only specific networking tasks to computing units (e.g., SoC-based SmartNICs). To support such a level of programmability, SmartNICs often rely on a wide variety of hardware platforms and programming languages (e.g., P4, Micro-C, VHDL/Verilog). Nevertheless, programming, debugging, and operating SmartNICs remain a challenging task.

Differently from programmable switches architectures (e.g., Tofino TNA architecture [Intel 2021]), SmartNICs usually follow a run-to-completion model where network packets are assigned to computing units without preemption [Guo et al. 2023]. Therefore, in multi-core SmartNICs, network packet processing may experience variable latency de-

[1] SmartNICs are also commonly touted as DPU (Data Processing Unit) or IPU (Infrastructure Processing Unit). Despite some attempts – mostly driven by marketing – to provide technical differences between a SmartNIC, DPU, and IPU, a well-accepted understanding is lacking. Therefore, and considering the scope and objectives of this book chapter, we use these terms interchangeably.

pending on the program structure, such as the number of Match/Action tables. That is not the case, for instance, in TNA architectures where resource constraints are a first-order concern, and performance guarantees come for free as long as the program fits inside the device.

To shed light on the design of emerging in-network solutions, this chapter introduces essential principles related to the design and operation of state-of-the-art SmartNICs. We cover and discuss different hardware architectures and the available programming software ecosystem. Then, we dive into the performance limitations of existing SmartNICs and discuss tailored networking applications for them. Finally, we present a hands-on tutorial with selected state-of-the-art SmartNICs (Nvidia BlueField, Mellanox ConnectX, and Netronome NFP).

2.2. Fundamentals of SmartNICs

This section provides a solid background on SmartNICs and on the Portable NIC Architecture (PNA) [(PNA) 2023]. We start with a broad overview of the network packet processing pipeline in existing NICs. Then, we will cover the state-of-the-art SmartNIC architectures: (i) Nvidia Bluefield, (ii) Xilinx Alveo, and (iii) Netronome NFP-4000.

As previously mentioned, NICs have evolved to support network speeds that go beyond 100 Gbps, while incorporating programmable units. The hardware architecture necessary for achieving high-speed network packet processing requires a high degree of parallelism to achieve performance scalability in NIC programs. To address this, current programmable NICs rely on multiple hardware architectures including (i) ASIC (e.g., Netronome NFP); (ii) System-on-Chip (e.g., Nvidia BlueField); and (iii) FPGA (e.g., Xilinx Alveo). Our discussion in this section will focus on the referred architectures, which play a central role in many academic studies [Liu et al. 2019, Min et al. 2021, Schuh et al. 2021, Wei et al. 2023a].

SmartNICs typically employ an alternative processing approach in comparison to an ASIC switch (e.g., Tofino), wherein a packet is directed to a specific processing engine following a run-to-completion model. In this model, a packet is assigned to a processor (or specific hardware) which executes all the instructions required for processing the packet. For example, Nvidia BlueField adopts a "disaggregated RMT" architecture [Chole et al. 2017]. In this design, a group of ASIC packet engines handles header computation and retrieves Match/Action (MA) entries from SRAM via a memory bus. In contrast, Netronome Agilio utilizes a collection of SoC-based CPU cores for packet processing, with corresponding entries situated in a more distant memory hierarchy [Xing et al. 2023]. In the case of multicore SmartNICs, packet latency can vary based on the program structure, including factors such as the number of Match/Action tables and their match types. Additionally, packets following distinct execution paths within the same program may encounter varying levels of latency. Despite that, the run-to-completion model is more flexible in the sequence of actions executed on the packet because a processor is not limited in the sequence of actions. This is not the case, for instance, in an RMT pipeline model (e.g., Tofino TNA architecture [Intel 2021]), where the sequence and number of used stages limit the programability.

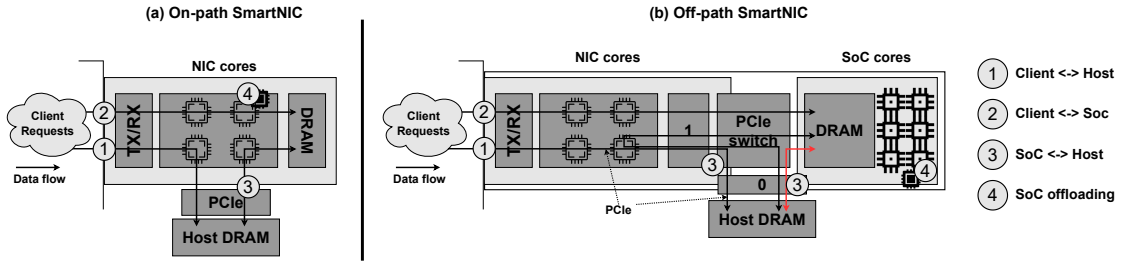


Figure 2.1. Overview of different SmartNIC architectures. Adapted from [Wei et al. 2023b].

2.3. SmartNIC Architectures

In this section, we discuss the main SmartNIC architectures in use on commercial programmable NICs. We start discussing general architectural designs: *on-path* and *off-path*. Then, we deep dive into System-on-Chip (SoC), Field Programmable Gate Arrays (FPGA), Application-Specific Integrated Circuit (ASIC), and hybrid architectures.

2.3.1. On/Off-Path Designs

SmartNICs can be categorized in *on-path* and *off-path* [Wei et al. 2023b]. On-path SmartNICs expose the NIC cores to the system with low-level programmable interfaces. In this design, the offloaded code is on the critical path of the packet processing pipeline and competes for the NIC resources. Therefore, if offloading too much computation, the NIC might suffer performance degradation. Also, NIC's core has direct access to memory subsystems such as DRAM or caching. Figure 2.1 illustrates the On-path design.

In turn, on *off-path* designs, NICs are equipped with additional computing cores and memory in a separate SoC, sitting next to the NIC cores. In this design, the offloaded code is off the critical path of the packet processing pipeline. In this case, the offloaded computation does not affect packet processing performance as long as it does not involve networking communication. Figure 2.1 illustrates the *off-path* design. Observe the computing cores are sitting next to NIC cores, interconnected by an internal PCIe interconnection.

2.3.2. Representative SmartNICs Architectures

2.3.2.1. SoC-based Designs

SoC architecture refers to a microchip that integrates most or all components of a computer or other electronic system into a single integrated circuit. These components typically include a central processing unit (CPU), memory, input/output ports, and various peripheral interfaces such as USB, HDMI, and Ethernet. In the context of SmartNICs, SoC architectures have been used to allow network programmability inside the NIC. Netorking-based SoC is usually composed of multiple CPUs (e.g., ARM cores), integrated with high-speed RAM, and fixed-function accelerators (e.g., encryption and regex engines). On top of the architecture, there is usually a running Operating System that manages NIC resources. In other words, a SoC SmartNIC can be seen as an individual computing node inside the server.

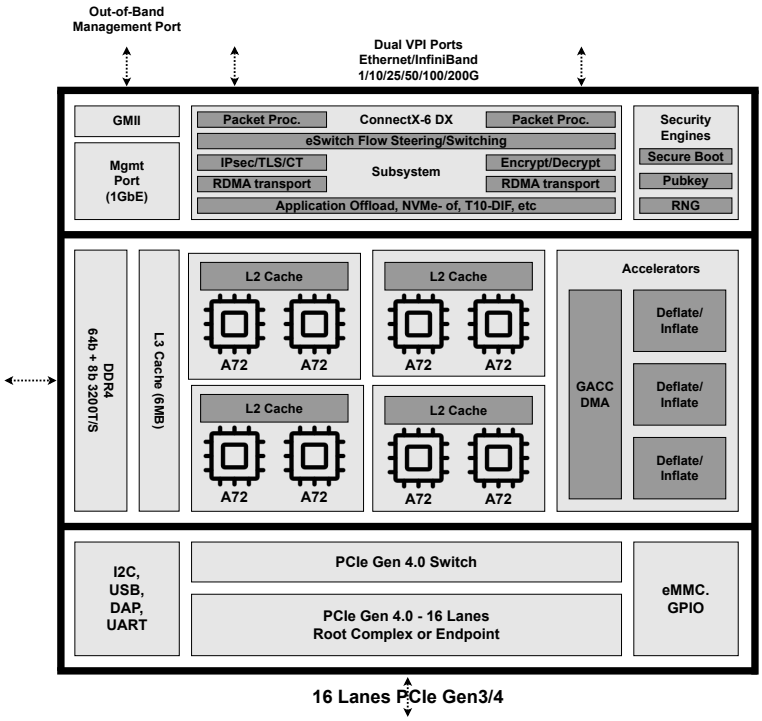


Figure 2.2. Overview of NVIDIA BlueField Architecture. Adapted from [NVIDIA 2024].

One representative SoC-based architecture is the Nvidia BlueField. Figure 2.2 illustrates the 100Gbit Nvidia BlueField-2 architecture. This SmartNIC has eight ARM cores interconnected in a mesh. ARM cores have direct access to DDR4 memory and specific networking accelerators (e.g., regular expression, SHA-2). That allows, for instance, to process packet payloads in hardware using the regular expression engine. One of the fundamental building blocks of this architecture is the eSwitch subsystem. Network packets are received from the Ethernet/Infiniband physical ports. Then, these network packets are processed by the eSwitch Flow Steering subsystem, which can be seen as a programmable switch inside the NIC (e.g., to Open vSwitch[Pfaff et al. 2015]). As such, it allows programming how network packets are steered from/to physical network ports, as well as to steer them to networking applications running on top of the ARM cores, or even send them via PCIe to the host CPU.

As there is an Operating System in the SoC-based SmartNIC, programmers can run any application inside it (ranging from high-performance DPDK-based applications to C-alike sockets). However, to get the best performance out of the architecture, each networking vendors provide a programming suite (e.g. DOCA framework) to better use the available resources (as we later explore in detail in Section 2.7). Therefore, different from a programmable switch (e.g., Tofino and its TNA architecture), SoC-based SmartNIC performance can vary depending on the running networking application. As there is no upper bound on the number of instructions each packet can be submitted to, the line rate (and the latency) can be compromised depending on how intense is the packet processing.

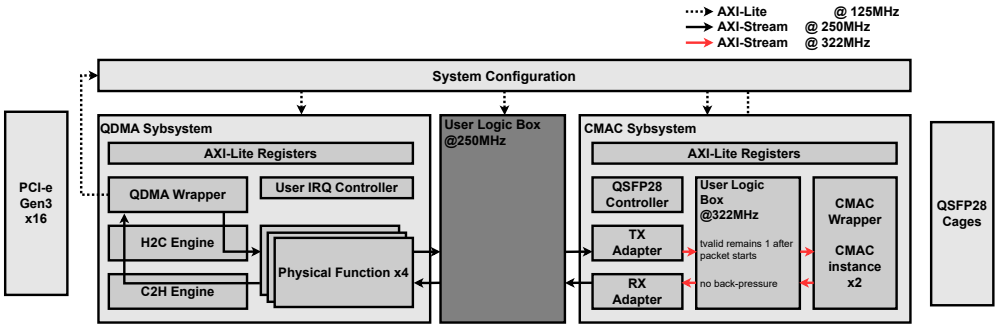


Figure 2.3. Overview of OpenNIC architecture. Adapted from [OpenNIC 2024].

2.3.2.2. FPGA-based Designs

An FPGA is an integrated circuit that allows users to configure it after manufacturing. Unlike conventional integrated circuits, which have fixed functionalities, FPGAs consist of programmable logic blocks and interconnects that can be customized to implement various digital circuits. By loading a configuration file onto the chip, users can specify how the logic blocks should be interconnected and what functions they should perform. This flexibility makes FPGAs suitable for prototyping, fast development, and applications requiring hardware reconfiguration or customization.

In the context of SmartNICs, there are a few vendors that provide FPGA-based SmartNIC (e.g. Xilinx). FPGA SmartNIC provides flexibility to prototype anything directly in hardware. However, the programmer needs to write the whole NIC hardware description (usually using High-Level Synthesis languages), and the operating system drivers. The NIC hardware is usually composed of many components (e.g., DMA subsystems, multiple RX/TX queues, Ethernet MAC subsystems, schedulers, etc). As one can observe, writing an FPGA hardware description might become a hard task to be done. Fortunately, the Open-Source community provides a reference FPGA NIC implementation called the OpenNIC project[2].

The OpenNIC project offers an FPGA-based NIC platform tailored for the open-source community. This platform comprises various components: a NIC shell, a Linux kernel driver, and a DPDK driver. The NIC shell encompasses RTL sources and design files, optimized for deployment on numerous boards featuring UltraScale+ FPGAs. It furnishes a NIC implementation supporting up to four PCI-e physical functions (PFs) and two 100Gbps Ethernet ports. The shell features well-defined data and control interfaces, facilitating seamless integration of user logic. Figure 2.3 illustrates the basic components of OpenNIC project.

2.3.2.3. ASIC-based Designs

ASIC-based SmartNICs utilize customized fixed-function processors (or integrated circuits) to process packets. One representative ASIC-based architecture is the Netronome

[2]`https://github.com/Xilinx/open-nic`

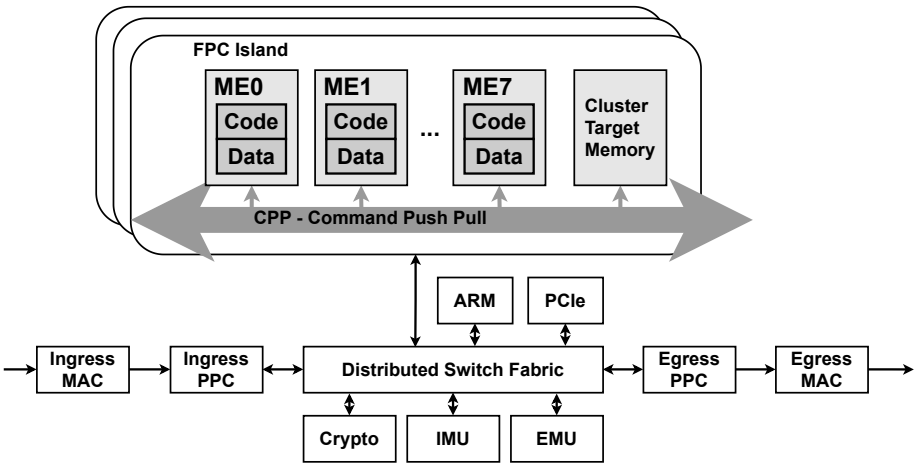


Figure 2.4. Overview of the Netronome architecture. Adapted from [Cannarozzo et al. 2024].

NFP4000. The architecture of the Netronome NFP4000 SmartNIC is depicted in Figure 2.4, with its flow processing cores (FPC) organized into multiple islands.

Each FPC operates as a 32-bit machine, with all internal registers and local memory consisting of 32-bit words. Within each FPC, there are eight Micro Engines (MEs), each functioning as a separate processor with its own instruction store (*code*) and local memory (*data*). Consequently, each ME can execute in parallel with all other MEs. With 8 threads per ME, cooperative multithreading is enabled, allowing at most one thread to execute code from the same program concurrently. Thus, each FPC can accommodate up to 8 parallel threads, running at a frequency of 1.2GHz (one thread per ME). FPCs adhere to a Harvard Architecture, where code and data are stored in separate memories: 4K bytes are shared between threads for data and private memory, while 8K instructions are reserved for the code store.

The local memory within each FPC consists of 32-bit registers shared among all 8 threads, categorized into: (i) general-purpose registers (256 registers of 32 bits each), (ii) transfer registers (512 registers of 32 bits each) for interconnection bus operations, (iii) next-neighbor registers (128 registers of 32 bits each) mainly for communication with neighboring FPCs, and (iv) local memory (1024 registers of 32 bits each), slightly slower than general registers with a 3-cycle access time. When local FPC registers cannot accommodate the required memory, variables are automatically and statically allocated to other in-chip memory hierarchies.

In addition to local memory, FPCs have access to four other types of memory: Cluster Local Scratch (CLS) memory (20-50 cycles), Cluster Target Memory (CTM) (50-100 cycles), Internal Memory (IMEM) (120-250 cycles), and External Memory (EMEM) (150-590 cycles). Each memory type serves specific purposes, ranging from storing frequently used data to managing packet headers and accommodating large shared tables.

Incoming packets from the network are picked up by an FPC thread from the Distributed Switch Fabric and processed accordingly, thereby constituting an on-path Smart-

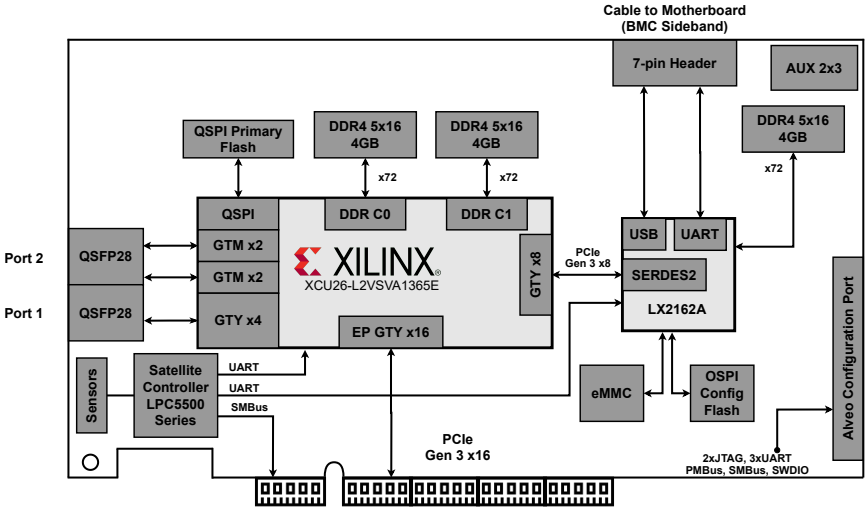


Figure 2.5. Overview of the Xilinx SN1000 architecture. Adapted from [Xilinx 2024].

NIC operation. For instance, the SmartNIC NFP-4000 supports up to 60 FPCs, with each FPC capable of handling up to 8 threads, enabling the device to process up to 480 packets simultaneously.

2.3.3. Hybrid Designs

In addition to the aforementioned SmartNIC architectures, there are still hybrid architectures made of a combination of existing architectures. The most popular combination is the SoC + FPGA architecture. These architectures provide the hardware expressiveness of FPGAs, combined with the flexibility of using ARM cores in the SoC. Note, however, that other architectural combinations also exist such as SoC + ASIC.

Figure 2.5 illustrates the architecture of Xilinx SN1000. As one can observe, the physical ports (QSFP28) are directly interconnected with the FPGA board (XCU26). The FPGA is directly connected to memory multiple subsystems and to the SoC subsystem (LX216A). The SoC in this architecture is composed of 16 ARM cores which also access to private memory subsystem.

2.3.4. Portable NIC Architecture (PNA)

Over the last years, the P4 (Programming Protocol-independent Packet Processors) language [Bosshart et al. 2014a] has been mainly used to program programmable switches. A P4 is an open source, domain-specific programming language for network devices, specifying how data plane devices process packets. The data plane architecture describes the structure and capabilities of the data plane device and exposes architecture-dependent functions to a P4 program [(PNA) 2023] (e.g., hashing functions). The P4 architectural reference (i.e., V1model[3]) reflects the pipeline nature of existent packet processing switches. However, with the emergence of other programmable network devices such as

[3]P4 reference model: `https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4`

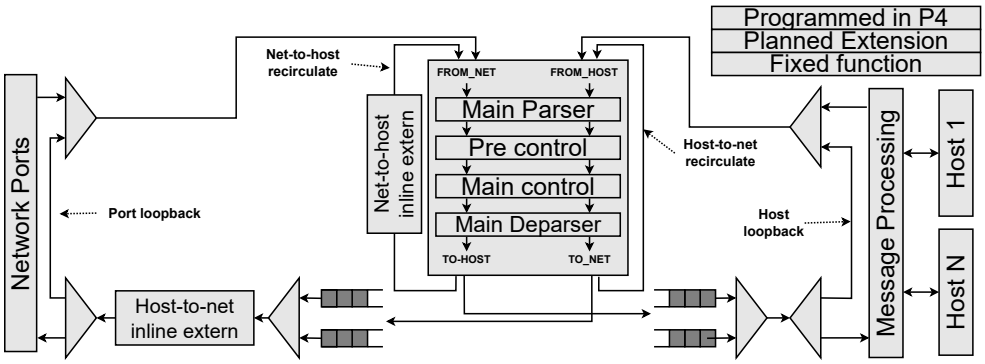


Figure 2.6. Overview of packet paths in the PNA architecture. Adapted from [(PNA) 2023].

SmartNICs, there is a need to adapt that architectural reference to fit existing new requirements/constraints.

In this context, the Portable NIC Architecture is a design framework aimed at encapsulating the fundamental characteristics of a wide array of Network Interface Controller (NIC) devices [(PNA) 2023]. It constitutes a P4 architecture delineating the organization and shared functionalities inherent to NIC devices. PNA consists of two primary elements:

1. A configurable pipeline engineered to accommodate diverse packet pathways traversing between different ports on the device, such as network interfaces or the host system to which it connects.

2. A library of types (e.g., intrinsic and standard metadata) and $P4_{16}$ externs (e.g., counters, meters, and registers).

The PNA Model incorporates four programmable P4 blocks and several fixed-function blocks, depicted in Figure 2.6. The operational characteristics of the programmable blocks are specified using the P4 language [Bosshart et al. 2014b]. Conversely, the network ports, packet queues, and potentially present inline extern blocks are categorized as fixed-function blocks, subject to configuration by the control plane, albeit not intended for programming via P4.

2.3.4.1. PNA Overview

As illustrated in Figure 2.6, network packets can have multiple paths in the context of a NIC. Different from a traditional switch-based P4 architecture (e.g., V1model, TNA, etc), where packets are coming/going to physical ports, in the PNA architecture packets can come from physical network ports, as well as to/from host interfaces. In the context of PNA architecture, these paths are referred to as (i) Net-to-Host, (ii) Host-to-Net, (iii) Net-to-Net, and (iv) Host-to-Host paths.

Packets arriving from a network port go through a MainParser and a PreControl building blocks. The MainParser, similar to other P4 architectures, is intended to parser network packets according to packet headers. For instance, a (Ethernet + IP + TCP) packet is going to be parsed according to the definitions of these three protocols. At the end of the parser process, the header fields of these protocols are stored in data structures that are available throughout the whole packet processing pipeline. Then, in the PreControl block, packets can optionally perform table lookups. Its purpose is to determine whether a packet requires processing by the net-to-host inline extern block. An example of an inline extern block that packets might be submitted to in the Net-to-Host path is the decryption of packet payloads according to the IPsec protocol. The PreControl code might drop the packet if the packet had an IPsec header, but one or more P4 table lookups determined that the packet does not belong to any security association that had been established by the control plane.

The MainControl is typically where the code would be written for processing packets. It transforms headers, updates stateful elements like counters, meters, and registers, and optionally associates additional user-defined metadata with the packet. The MainDeparser serializes the headers back into a packet that can be sent onward. After the MainDeparser, the packet may either: proceed to the message processing part of the NIC, and then typically on to the host system, or turn around and head back towards the network ports. This enables on-NIC processing of port-to-port packets without ever traversing the host system.

While the primary programmable blocks focus on handling individual network packets, typically limited to a single network maximum transmission unit (MTU) in size, the message processing block assumes the responsibility of facilitating the conversion between large messages stored in host memory and network-sized packets for transmission across the network. Moreover, it manages interactions with one or more host operating systems, drivers, and/or message descriptor formats residing in host memory.

For instance, when converting large messages to network packets in the host-to-network direction, the message processing block orchestrates functionalities such as Large Send Offload (LSO), TCP Segmentation Offload (TSO), and Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE). Conversely, in the network-to-host direction, it aids in features such as Large Receive Offload (LRO) and RoCE.

2.3.4.2. Differences from P4$_{16}$ language

Although the PNA architecture relies on the P4$_{16}$ language definition, some new features are still not defined in the base P4 language. In particular, there are mach-action table properties that are not included in the base P4$_{16}$ language specification.

The *add_on_miss* table property is defined by the PNA. When this property is set to true for a table, P4 developers can specify a default action for the table, which invokes the *add_entry* extern function. This function adds a new entry to the table, where the default action involves calling the *add_entry* function. The newly added entry will possess the identical key that was recently queried. That allows to create, for instance, self-contained data plane applications that do not depend on the control plane. The control

plane API would still be able to add, change, and remove entries from the mentioned table. However, entries added through the *add_entry* function operate independently of the control plane software.

Another table property envisaged in the PNA architecture is the *table entry timeout notification*. PNA utilizes the *pna_idle_timeout* parameter to allow a table implementation in sending notifications from the PNA device. These notifications are triggered when a configurable duration has elapsed since an entry was last matched.

2.3.4.3. Existing efforts to make PNA usable

Despite the growing involvement of the Open-Source community to make PNA a standard architecture for SmartNICs, little is still possible to be done using PNA to program SmartNICs. To date, the only P4 backend compiler that adheres to the PNA architecture is the `p4c-dpdk`[4]. The `p4c-dpdk` backend translates the P4-16 programs to DPDK API to configure the DPDK software switch (SWX) pipeline. DPDK introduced the SWX pipeline in the DPDK 20.11 LTS release. Each pipeline is created using a specification file that can either be manually developed or generated using a P4 compiler.

Therefore, one can write P4-based PNA code, compile it to DPDK, and run it either on SmartNICs (e.g., SoC-based ones) or in the host system using DPDK-enabled NIC interfaces using the DPDK pipeline application. For more information, please refer to the official DPDK documentation [DPDK 2024].

2.4. SmartNIC Software & Hardware Ecosystem

In this section, we provide an overview of SmartNICs from the vendor's perspective. We discuss the available SmartNIC models considering the existing architectures, the programming suite available, and existing hardware requirements and specifications.

2.4.1. Brief Historical Perspective

The emergence of the SmartNIC term dates back to the 90s [Ponomarev and Ghose 1998]. However, at that time, NICs were primarily responsible for interfacing a computer with a network. They handled basic functions such as packet reception, transmission, and some basic processing tasks like checksum offloading.

NIC offloading has evolved from basic functionality to more advanced and specialized offloading capabilities, driven by the increasing demands of modern networking environments and the need for improved performance, efficiency, and scalability. For instance, in the 2000's, NICs have been empowered with TCP Offload Engine. Later, in the 2010's, NICs evolved to support virtualization-driven requirements such as VLAN, RSS, etc.

However, it was only after the emergence of Software-Defined Networking (SDN) (and later the P4 language) concepts that SmartNICs started to emerge as a "smart" device. The network programmability fostered by these concepts has driven the development of

[4]`https://github.com/p4lang/p4c/blob/main/backends/dpdk/README.md`

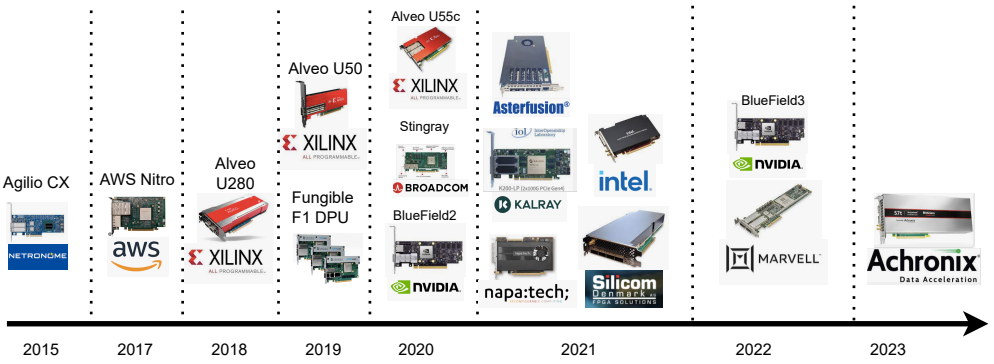


Figure 2.7. SmartNIC vendors timeline.

high-level programmable NICs. Figure 2.7 illustrates SmartNIC vendors over the last year. Note that this is not an exhaustive list and many models are not mentioned.

In 2015, Netronome[5] was the first vendor to announce a SmartNIC model supporting data plane user-level programmability using P4 language. They provided a P4-to-MicroC compiler, allowing the user to write their own NIC firmware with full access to the NIC computing power. Then, in 2017, Cloud players also announced the design of SmartNICs to support the ever-growing demands of their data centers (e.g., AWS Nitro). In 2018, Xilinx announced the Alveo SmartNICs designed also aimed at data center workloads. These cards were one of the first FPGA-based SmartNICs. These cards leverage Xilinx's FPGA technology to accelerate a wide range of compute-intensive tasks. In 2019 and 2020, SoC-based SmartNICs were announced such as the NVIDA BlueField. Following this trend, in 2021, Intel also announced Intel SoC interfaces. More recently, Marvel and Asterfusion also announced powerful SoC-based SmartNICs with up to 36 ARM cores. Table 2.1 summarizes the main SmartNICs by vendor.

As we observe in Table 2.1, SmartNICs can handle a few hundred gigabytes per second. In 2015, Netronome architecture was able to handle up to 40Gbit/s. Today, we see SmartNICs supporting up to 200Gbit/s per physical port. SmartNIC models are evolving together with just approved Ethernet standards. The IEEE P802.3bs Task Force developed the 400 Gigabit Ethernet (400G, 400GbE) and 200 Gigabit Ethernet (200G, 200GbE) standards, which were approved back in 2017. These standards employ technology similar to that of 100 Gigabit Ethernet. Last, in 2024, the IEEE P802.3df Task Force approved the 800 Gigabit Ethernet (800G, 800GbE) standard.

2.4.2. Programming SmartNICs

Designing and implementing SmartNIC programs might take time and effort, depending on the vendor and available architecture. Despite increasing interest in PNA architecture and P4 language, they are still very restricted and are not widely available. For this reason, each vendor/model takes advantage of its development framework. Next, we overview the main programming frameworks available from the leading vendors: Netronome, Xilinx, and NVIDIA. We later deep dive into the NVIDIA framework as it is used in Section 2.7

[5]https://netronome.com/

Table 2.1. An overview of SmartNIC ecosystem.

Vendor	Model	Release Year	Architecture	Bandwidth	PCIe	Programming Suite	Features
Achronix	Speedster7t FPGA series	2023	FPGA	400 Gbit/s	PCIe 5.0	Achronix Tool Suite (Verilog/VHDL)	Achronix FPGA 692K LUTs
Napatech	NT200A02	2021	FPGA	100Gbit/s	PCIe 3.0	Vivado Suite	Xilinx FPGA UltraScale+
Xilinx	u280	2018	FPGA	100Gbit/s	PCIe 4.0	Vivado Suite	Xilinx FPGA UltraScale+ 1079K LUTS
Xilinx	u55c	2020	FPGA	100Gbit/s	PCIe 4.0	Vivado Suite	Xilinx FPGA UltraScale+ 1079K LUTS
Intel	N5010	-	FPGA	4x 100Gbit/s	PCIe 4.0	Intel Open FPGA Stack (Intel OFS)	Intel Stratix 10 DX 2073K LUTS
Intel	N6000-PL	2021	FPGA	200Gbit/s	PCIe 4.0	-	Intel FPGA Agilex 7 FPGAs F-Series 1437k LUTS
Asterfusion	Helium EC2002P	2021	SoC	200Gbit/s	PCIe 3.0/PCIe 4.0	DPDK/VPP Development Kit	24 ARM 1.8Ghz + Dedicated acellerator + ASIC
Fungible/Microsoft	F1 DPU	2019	SoC	800Gbit/s	4x PCIe 4.0	P4	MIPS-64, 9-stage, dual-issue, 4xSMT, FPU/SIMD unit
Kalray	K200/K200-LP	2021	SoC	100Gbit/s	PCIe 4.0	P4	5 cluster of 16 cores – MPPA – specific cores – 600 to 1.2 Ghz
Marvell	LiquidIO III	2022	SoC	2x 50Gbit/s	PCIe 4.0	DPDK, VPP, SPDK	Up to 36 cores ARM V8 at 2.2GHz
NVDIA	BlueField-2	2020	SoC	2x 100Gbit/s	PCIe 4.0	DOCA framework, DPDK	8 Armv8 A72 cores
NVIDIA	BlueField-3	2022	SoC	400Gbit/s	PCIe 5.0	DOCA framework, DPDK	16 Armv8.2+ A78 Hercules cores
Broadcom	Stingray	2020	SoC	100Gbit/s	PCIe 4.0	DPDK	cluster of four 3 GHz dual-core Arm v8 A72 complexes
Intel	Intel FPGA IPU F2000X-PL Platform	2021	SoC + FPGA	200Gbit/s	4x PCIe 4.0	IPDK, SPDK P4 Programmable	Intel Agilex 7 FPGA F-Series 2300k LUTS + 8-core Intel Xeon-D SoC
Xilinx	SN1000	2021	SoC+FPGA	200Gbit/s	PCIe 4.0	Vivado, Vitis, P4, DPDK	AMD XCU26 FPGA Ultra-Scale+, 16-core Arm
Netronome	Agilio CX	2015	ASIC	40Gbit/s	PCIe 3.0	P4, Micro-C / Agilio Software	48/60 NFP-4000 flow processor

in a step-by-step, hands-on programming tutorial.

2.4.3. ASIC-based SmartNIC: Netronome Programming Framework

The Netronome was the first vendor to support P4 language as the primary way to program the SmartNIC data plane. Netronome supports $P4_{14}$ and $P4_{16}$, however much of the development framework is still limited to $P4_{14}$. That includes, for instance, the Netronome simulation engine.

The P4 development in the Netronome board follows a similar architecture to the v1model reference. This allows programmers to translate P4 code seamlessly into the SmartNIC architecture. Besides programming the SmartNIC using P4 language, Netronome allows users to write low-level Micro-C code. Micro-C can be written as P4 externs or as a standalone data plane application. This way, users can get the best out of the Netronome hardware architecture. For instance, Netronome P4 compiler assumes the code is identically running in all NFP cores. Furthermore, it applies simple heuristic procedures to utilize the memory hierarchy (e.g., NFP caches, DRAM, etc). Therefore, the P4 programming might end up with data plane misbehavior for applications that utilize external memory extensively. In those situations, it is recommended to write Micro-C code instead of relying on P4 translations.

The Netronome SmartNIC is backed up by a Linux Run Time Environment (RTE) service that allows flexible interaction with the SmartNIC application from the operating system perspective. That allows, for example, populating Match-Action tables or creating more refined Control Plane applications. Unfortunately, OpenFlow or P4Runtime are not supported by default.

In addition to allowing the data plane firmware to be written from scratch, the Netronome also provides Linux drivers with native support for eBPF/XDP. Therefore, it also allows hardware programability when offloading eBPF/XDP instructions to be executed by the SmartNIC.

2.4.4. FPGA-based SmartNIC: Xilinx Programming Framework

Xilinx Vitis represents a pioneering advancement in development environments, spearheaded by Xilinx to redefine the dynamics of embedded software and hardware platform development. Within its unified interface, Vitis seamlessly converges software and hardware development tasks, accommodating a range of programming languages, including C, C++, OpenCL, and the P4 language. This comprehensive integration allows developers to leverage the unique capabilities of the P4 language, which is particularly advantageous for programmable data planes in networking applications, thereby enhancing both flexibility and performance in their projects.

Central to Vitis is its holistic suite of features, meticulously designed to simplify and elevate the development process. Developers can succinctly articulate algorithmic descriptions through high-level abstractions, while pre-built acceleration libraries cater to common tasks such as image processing and machine learning, optimizing performance and efficiency. The environment's inherent versatility extends to seamless integration with Vivado, Xilinx's acclaimed FPGA synthesis and implementation tool, ensuring a seamless transition from algorithmic formulation to hardware realization. Such synergies empower

developers to efficiently map their applications onto Xilinx hardware platforms, accelerating innovation across diverse domains, including artificial intelligence, high-performance computing, automotive, aerospace, and telecommunications.

Moreover, Xilinx offers flexible licensing options for Vitis, providing developers with tailored access to its comprehensive suite of tools and features. Whether through perpetual licenses or subscription-based models, developers can choose the licensing option that best aligns with their project requirements and budget constraints. With Vitis and its adaptable licensing structures, developers are equipped to confidently navigate the complexities of modern development landscapes, forging new frontiers in technology and driving forward the boundaries of possibility.

2.4.5. SoC-based SmartNIC: Nvidia Programming Framework

NVIDIA DOCA brings together a wide range of powerful APIs, libraries, and frameworks for programming and accelerating modern data center infrastructures. DOCA is a consistent and essential resource across all existing and future generations of BlueField DPUs. DOCA offers a rich set of APIs that allow interactions with the BlueField computing units. These include DOCA Flow, DOCA Core, DOCA RDMA, and DOCA GPUNetIO, among others. For a complete list of APIs, please consult the official documentation[6]. In this chapter, we overview the DOCA Flow API, as this is used later for a hands-on tutorial.

DOCA Flow is the most fundamental API available, as it enables the creation of generic packet processing pipes in hardware. The DOCA Flow library provides an API for constructing a set of pipes, each consisting of match criteria, monitoring, and a set of actions. Pipes can be chained so that after a pipe-defined action is executed, the packet may proceed to another pipe. A pipe is a template that defines packet processing without adding any specific hardware rule. It comprises four elements: Match, Monitor, Actions, and Forward.

Figure 2.8 illustrates a pipeline implementation in DOCA Flow. On using DOCA Flow, it is easy to develop hardware-accelerated applications that have a match on up to two layers of packets: (i) MAC/VLAN/ETHERTYPE, (ii) IPv4/IPv6, (iii) TCP/UDP/ICMP, (iv) GRE/VXLAN/GTP-U, or (v) packet metadata. The execution pipe can also have monitoring actions such as counters and policers. Last, the pipe also has a forwarding target: software application (RSS to subset of queues), physical/virtual port, another pipe, and Drop packets.

DOCA Flow pipes offer a versatile framework with a user-defined set of matches parser and actions, allowing for precise control over packet processing. These pipes can dynamically create or destroy, adapting to changing network demands seamlessly. Leveraging specialized hardware acceleration, packet processing within these pipes achieves optimal efficiency. Each flow pipe contains specific entries tailored to accelerate packet handling, ensuring high-performance throughput. In cases where packets fail to match any hardware entries, Arm cores step in for exception handling, providing a robust mechanism to address diverse scenarios. Following exception handling, packets are seamlessly

[6]`https://docs.nvidia.com/doca/`

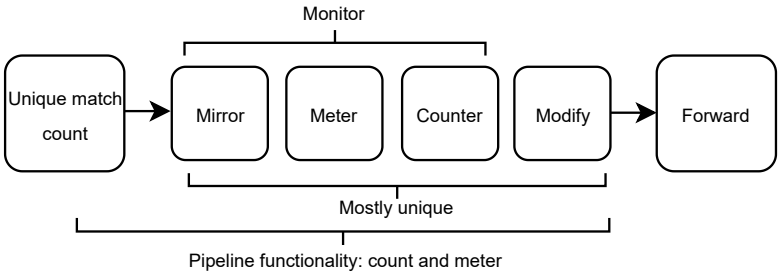


Figure 2.8. DOCA Flow Pipeline. Adapted from NVIDIA DOCA documentation [NVIDIA 2024].

re-injected back into the hardware pipeline, maintaining the flow's integrity and efficiency.

2.5. Performance Benchmarking

This section focuses on the details of performance benchmarking designed specifically for certain models of SmartNICs. We examine important concepts, methodologies, and key considerations from recent academic studies that have benchmarked some SmartNIC models.

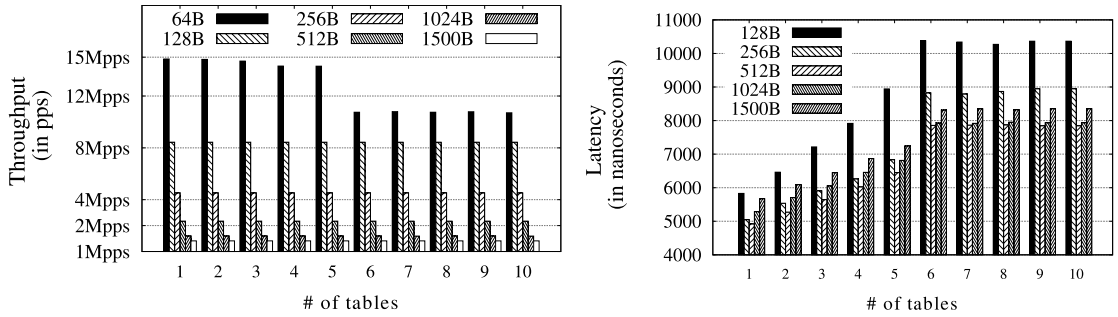
It is worth noting that, despite the growing popularity of SmartNICs, there have been relatively few efforts to benchmark these hardware devices. Furthermore, benchmarking is usually conducted individually and does not involve comparing multiple models or architectures. Nevertheless, we hope that this summary of existing benchmarks can provide an overview of the current performance of various SmartNIC models. In the following, we provide information about some of the performance evaluations available in the literature. These evaluations cover relevant aspects of the current performance and bottlenecks of SmartNICs, including the Netronome NFP4000 SmartNIC and the SmartNICs from the Mellanox ConnectX family.

2.5.1. Performance Evaluation of ASIC Netronome NFP4000 SmartNIC

As previously mentioned, the Netronome NFP4000 represented a pioneering advancement in the domain of SmartNICs, characterized by its sophisticated architecture and versatile programmability. In the study of Viegas et al. [Viegas et al. 2021], they provide detailed benchmarking of the performance of P4 instructions running on the NFP4000. To do this, they evaluate the cost (in terms of latency and throughput) of the main operations available in P4, in addition to varying SmartNIC parameters, such as the use of Micro Engines (or processing units). Their evaluation covered aspects such as tables, register operations, recirculations, hash functions, ingress and egress pipelines, and packet sizes. Next, we summarize their SmartNIC evaluation for table accesses, register operations, and recirculations, as we consider the main operations in current P4 programs.

First, we discuss the results for Match-Action table accesses in P4 programs. In the context of SmartNICs, this evaluation is important because, unlike conventional forwarding devices that utilize Match-Action tables solely for routing purposes, the P4 language allows for more diverse applications of this construction. The authors aim to

analyze how using multiple match-action tables at different pipeline stages affects performance. In the experiments, they vary the number of tables in their P4 programs and ensure sequential matching on all tables for every packet. Upon packet matching, an action is invoked to read a single 32-bit data from the table and store it in a metadata structure. Packet size and the number of tables per pipeline (ingress or egress) also vary. Figure 2.9 illustrates the measured throughput 2.9(a) and latency 2.9(b). Throughput behavior shows that for small packets, the throughput remains nearly constant for up to 5 Match-Action tables before experiencing a sharp drop. The Netronome architecture limits a P4 program to 5 tables in each pipeline. Interestingly, the ingress pipeline consistently demonstrates faster memory allocation, even when tables are defined only in the egress pipeline.



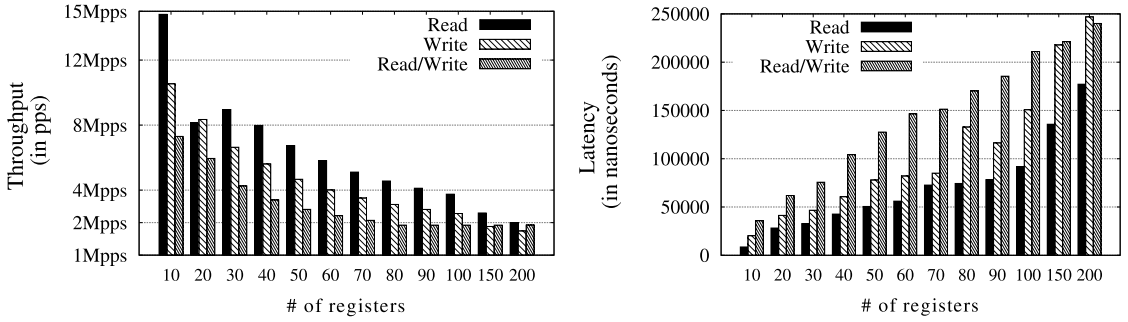
(a) Measured throughput cost.

(b) Measured latency cost.

Figure 2.9. Table access costs in the P4 pipeline. Figure from: [Viegas et al. 2021]

To evaluate the memory access costs (register access in P4 code), the authors examine the cost associated with executing multiple register operations within the same P4 pipeline. Register operations serve as fundamental components in modern P4 applications. Throughout the experiments, they varied the number of register operations conducted consecutively by the P4 program, ranging from 10 to 200 register operations, keeping the register width 32-bit. The authors specifically investigate the placement of registers within the ingress pipeline, categorizing them as read-only, write-only, or read-and-write operations. Figures 2.10(a) and 2.10(b) show throughput and latency, respectively. With increasing registers and P4 instructions, significant performance degradation occurs. Line rate sustains only with 10 registers for reading operations, but even with 10 registers, writing operations experience a 30% bandwidth drop (50% for read & write). Throughput linearly decreases by up to 87% (i.e., 2 Mpps) with 200 registers. Latency increases linearly with operation count per packet, from acceptable levels (e.g., 10 registers: 8650ns reading) to higher levels (e.g., 50 registers: 0.12 milliseconds).

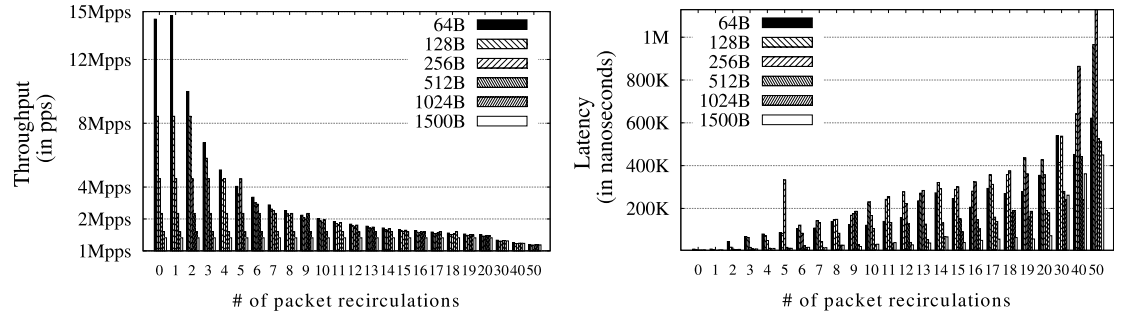
Finally, we overview their evaluation of the implications of packet recirculation within the P4 pipeline. Given P4's lack of support for iteration-based structures, packet recirculation is a workaround to emulate loop-based functionalities. Packet recirculation involves sending a packet back to the ingress pipeline post-processing, mimicking a loop-based structure. Throughout the experiments, they vary the number of packet recirculations per packet, ranging from 0 to 50, alongside altering the packet size from 64B to 1500B. The focus lies on forwarding network traffic from physical interfaces. Figure 2.11 illustrates the measured throughput 2.11(a) and latency 2.11(b). Throughput behavior shows a super-linear decrease with an increase in packet recirculations. Fewer



(a) Measured throughput cost. (b) Measured latency cost.

Figure 2.10. Register operation costs in the P4 pipeline. Figure from: [Viegas et al. 2021]

recirculations sustain a line rate for small packets, while larger packets maintain a line rate even with multiple recirculations. As packets recirculate, more are pushed into the data plane, causing enqueueing and eventual drop, reducing throughput. Observed latency also increases notably as packets recirculate. Even for large packets, per-packet latency doubles with just three recirculations, with sharper increases as the number of recirculations rises, especially for small packets.



(a) Measured throughput cost. (b) Measured latency cost.

Figure 2.11. Packet recirculation costs in the P4 pipeline. Figure from: [Viegas et al. 2021]

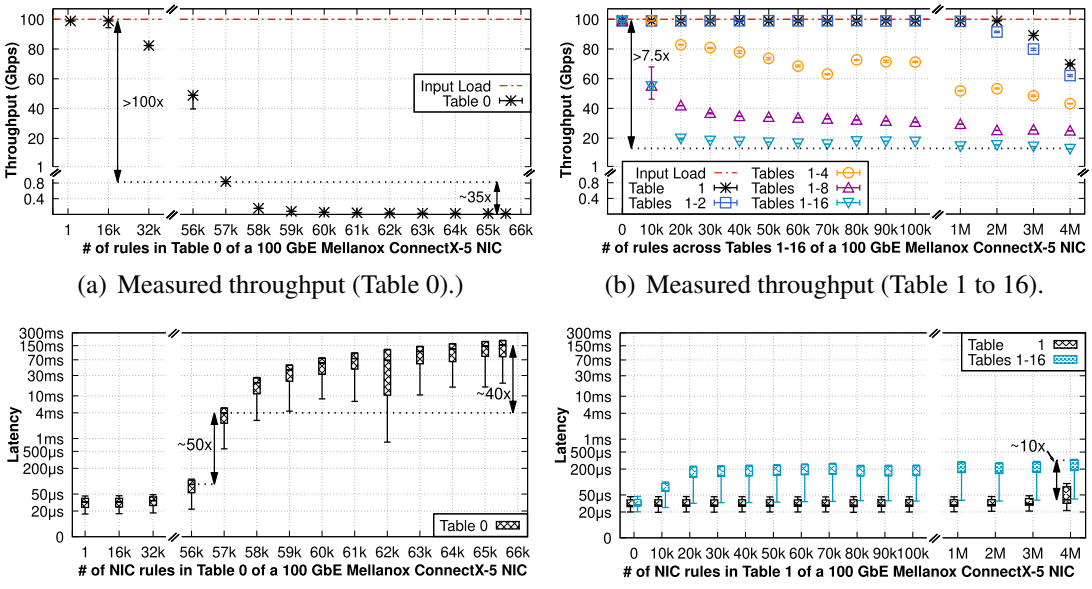
2.5.2. Performance Evaluation of ASIC/SoC Mellanox SmartNIC

The NVIDIA Mellanox ConnectX SmartNICs offer unparalleled networking performance and processing capabilities, operating at link speeds of up to 100 Gbps and beyond. Other vendors' models have certain limitations such as lower link speeds and limited flow rule capacities. For instance, the upcoming Intel E810 series still needs to catch up when compared to the ConnectX NICs. This demonstrates the superior scalability and performance capabilities of Mellanox SmartNICs. Next, we present a performance evaluation of some ConnextX SmartNICs, providing valuable insights into their performance evolution and highlighting the significance of ConnectX SmartNICs in modern data center environments.

The evaluation carried out by Katsikas et al. [Katsikas et al. 2021] delves into the architectural intricacies and differences among the NVIDIA Mellanox NICs, emphasizing their robust capabilities and innovative design features. The authors evaluate the

ConnextX-4, ConnextX-5, ConnextX-6 and BlueField SmartNIC. Leveraging a PCIe 3.0 x16 bus interface, except for the ConnectX-6 adapter, which utilizes two PCIe 3.0 x16 slots, these SmartNICs ensure optimal connectivity and throughput to the server's CPU. Furthermore, including an 8-core ARM processor in the BlueField-2 (and a 16-core ARM processor in BlueField-3) NIC extends its capabilities for in-NIC traffic processing, enhancing overall network performance and efficiency. Notably, Mellanox NICs boast a root table with ample space for rule entries and high-performance exact-match tables that facilitate efficient packet classification and offload tasks from the CPU. With the flexibility to accommodate many rules limited only by the host's available memory, Mellanox SmartNICs emerge as indispensable components in modern server architectures, offering unmatched scalability, performance, and flexibility for demanding networking environments.

In their evaluation, the authors cover aspects such as the cost of the number of table rules pre-installed, the impact of batch and rate-based updates, the impact of inserting new rules, and the impact of in-memory or out-memory updates. In our summary, we only present the cost of the number of rules pre-installed and the cost of rate-based updates, as they are the main table operations. Their experiments use a single-core forwarding network function on a Device Under Test (DUT). The DUT's NIC routes incoming frames to this network function based on flow rules stored in the NIC. These rules are in either the default "root" flow table or non-root tables. The results presented an overview of the ConnectX-5 NIC results, but the observed trends apply to other NICs tested.



(a) Measured throughput (Table 0).)

(b) Measured throughput (Table 1 to 16).

(c) Measured latency (Table 0).

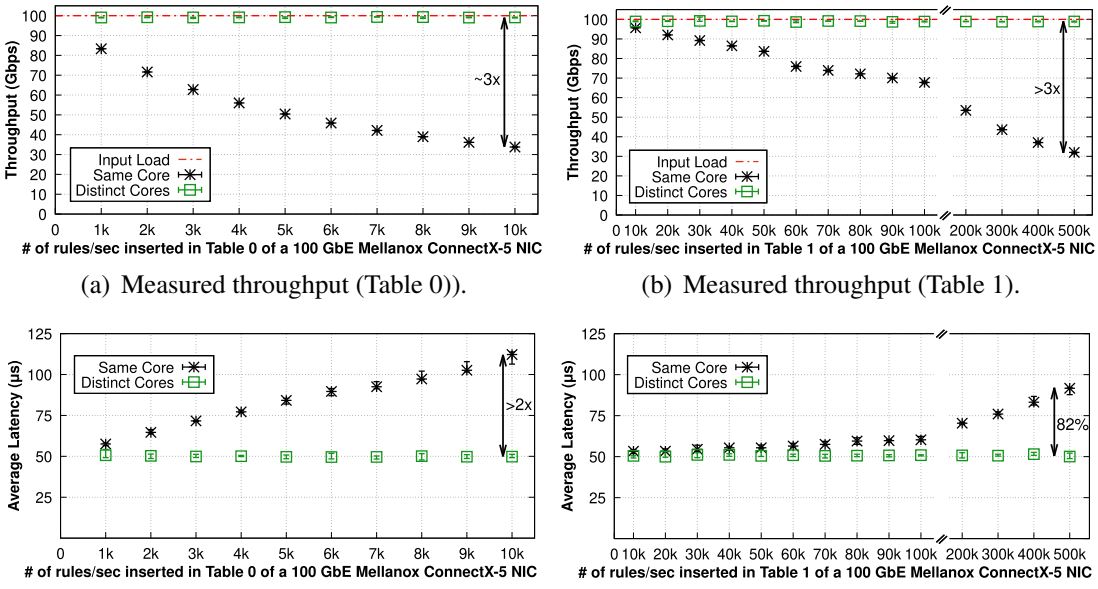
(d) Measured latency (Table 1 to 16).

Figure 2.12. Throughput and latency costs for a different number of pre-installed table rules. Figure from: [Katsikas et al. 2021]

Figure 2.12 illustrates how the performance of packet classification varies with the number of rules installed on different types of tables in the NVIDIA Mellanox ConnectX-5 NIC. When rules are uniformly distributed across non-root tables, throughput remains stable even with many entries. However, performance dramatically declines when the

root table (Table 0) reaches over 85% occupancy, rendering the last 15% of memory practically unusable. Despite lower input loads, throughput decreases significantly, indicating a design issue with the root table rather than an excessive load. Additionally, while non-root tables exhibit faster throughput and lower latency, spreading rules across more tables leads to performance degradation, highlighting trade-offs in table distribution strategies.

Next, the authors evaluate the cost of updating rules in a rate-based way. Periodic rule installations from a single core are typical in systems like NATs and Layer 4 load balancers. Figure 2.13(a) and Figure 2.13(b) illustrate forwarding network function throughput during simultaneous rule insertions into the NIC classifier. When insertions originate from a different core, throughput remains stable. However, using the same core as the forwarding network function results in a notable performance drop, with throughput decreasing by approximately 70 Gbps for 10K and 500K rule insertions per second in Table 0 and Table 1, respectively. This highlights a bottleneck in the NIC's standard API for updating the forwarding table. While using a different core for rule installation helps, it requires costly inter-core communication and consumes significant CPU resources, such as 100% of a core for several hundreds of milliseconds to install 500K rules. Latency increases by more than 2x for Table 0 and 82% for Table 1 when rule insertions occur from the same core, indicating significant performance degradation due to interference in the NIC data plane.



(a) Measured throughput (Table 0)).

(b) Measured throughput (Table 1).

(c) Measured latency (Table 0).

(d) Measured latency (Table 1).

Figure 2.13. Throughput and latency costs for rate-based updates in table rules. Figure from: [Katsikas et al. 2021]

2.6. Related Work

In this section, we describe recent works that used the SmartNICs presented in the previous sections to optimize or accelerate network functions. To do this, we initially present Table 2.2, which contains works published in the last four years (i.e., within 2020-2023) in some of the high-impact conferences that used SmartNICs in their solutions or exper-

iments. In our search, we consider the ACM SIGCOMM (ACM Special Interest Group on Data Communication)[7], USENIX NSDI (Symposium on Networked Systems Design and Implementation)[8], and the ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)[9] conferences, and the table presents the title of the work, conference and year, model, and architecture of the SmartNIC used. Please note that this is not an exhaustive list of all the works published in these conferences. Still, we aim to provide a comprehensive overview of the most relevant recent research studies that highlight the relevance of SmartNIC architectures in high-impact research. There may be other works related to SmartNICs in other high-impact conferences and journals that we may have missed.

Next, in addition to the general overview provided in Table 2.2, we also discuss recent works that use SmartNICs in their solutions in more detail. We divided this description into subsections based on SmartNIC architectures and discussed three works for each architecture.

2.6.1. FPGA-based SmartNICs

The increasing reliance of data center applications on proxies that shift application layer processing into the network motivated the emergence of new packet dispatch strategies. However, prior work has shown that this comes with a performance and resource cost. Wang et al. [Wang et al. 2023] aim to address this challenge by investigating the possibility of offloading Layer 7 (L7) processing to hardware, specifically FPGA SmartNICs. They focus on L7 dispatch, which involves analyzing application requests and distributing them to target service processes. Unlike L3/4 processing, L7 logic operates on application data spanning multiple packets, making offloading more complex due to variable-length fields and packet reassembly. The authors introduce QingNiao, a solution comprising a co-designed protocol, application interface, and hardware design, aiming to offload L7 dispatch efficiently. Prototyped on an FPGA integrated with a 100Gbps Corundum NIC, QingNiao demonstrates programmability and performance gains over software-based L7 dispatch. Results indicate throughput improvements of 7.5x to 8x and latency reductions of 72.5% to 74% compared to state-of-the-art software solutions, showcasing the viability of hardware offloading for L7 processing. The authors have made their software and hardware designs open-source to facilitate further research in L7 processing offload.

Yao et al. [Yao et al. 2023] address challenges in network scheduling by introducing the Balanced Multi-Way sorting Tree (BMW-Tree). This novel data structure aims to improve the efficiency and scalability of scheduling algorithms, particularly in modern data centers with high flow volumes. The BMW Tree enables the realization of the Push-In-First-Out (PIFO) model, which is crucial for packet prioritization. The paper presents two hardware designs, register-based (R-BMW) and RPU-driven (RPU-BMW), leveraging the BMW-Tree concept. These designs offer high throughput and scalability, addressing the limitations of traditional implementations. Evaluation of the proposed hardware designs using Verilog targeting a Xilinx Alveo U200 Data Center Accelerator Card demonstrates their significant performance improvements over traditional PIFO im-

[7] https://www.sigcomm.org/events/sigcomm-conference
[8] https://www.usenix.org/conference/nsdi23
[9] https://conferences.sigcomm.org/co-next/2023/

plementations. For instance, an 11-level 2-way R-BMW achieves a throughput of 192 Mpps (million packets per second) for 4k flows, while an 8-level 4-way RPU-BMW supports 87k flows at 93 MHz.

In turn, Zhang et al. [Zhang et al. 2023] explore the challenges of federated learning (FL) and the cryptographic techniques used to secure cross-silo FL operations. It identifies nine common cryptographic operations and highlights the performance degradation. The study proposes the FLASH architecture for hardware acceleration, focusing on FPGA and ASIC implementations. Evaluation of FLASH demonstrates significant performance improvements over CPU and GPU implementations across cryptographic operations and FL applications. Specifically, FLASH outperforms CPU and GPU by 10.4x to 14.0x and 1.4x to 3.4x, respectively, across cryptographic operations. Additionally, software evaluation as an ASIC shows further performance gains of up to 23.6x and 7.1x over FPGA implementation with 12nm and 28nm fabrication techniques, respectively.

2.6.2. SoC-based SmartNICs

Olteanu et al. [Olteanu et al. 2022] investigate challenges in data center networks, proposing the Edge-Queued Datagram Service (EQDS) to improve network utilization and support diverse transport protocols. EQDS moves queuing from switches to network edges, providing a datagram service via dynamic tunnels. Its receiver-driven control loop manages inbound traffic, ensuring isolation between protocols and facilitating fair sharing. EQDS offers improved protocol performance, protection from queuing delays, and increased throughput through load balancing. The study details EQDS design, implementation, and evaluation on Linux hosts and BlueField-2 SmartNIC, demonstrating its compatibility with existing transport protocols and its ability to enhance network performance.

Wei et al. [Wei et al. 2023c] delved into the performance characteristics of various communication paths on off-path SmartNICs, shedding light on essential aspects often overlooked in prior research. It highlighted the increasing adoption of RDMA in modern data centers, driving network bandwidth towards 400 Gbps. However, the intensified network speed demands more CPU resources to leverage RDMA-capable NICs, significantly burdening distributed systems fully. While one-sided RDMA can mitigate CPU pressures by allowing direct host memory access, limited offloading capabilities may lead to network amplification and performance degradation. Amidst these challenges, the emergence of SmartNICs with programmable capabilities offers a promising avenue for offloading complex computations. Two main types of SmartNICs are distinguished: on-path SmartNICs, which expose NIC cores for direct processing, and off-path SmartNICs, which employ programmable multicore SoCs adjacent to the RNIC cores. Due to their generality and programmability, the study primarily focused on off-path SmartNICs, such as NVIDIA Bluefield-2 and Broadcom Stingray. The study found that the RDMA path from the NIC to the SoC can be up to 1.48 times faster than the path to the host. It also revealed that RDMA requests involving the SoC may suffer from up to 48% bandwidth degradation due to performance anomalies introduced by the SoC.

Today, computing environments require seamless integration of hybrid environments spanning edge, cloud, and HPC systems, connecting sensors, elastic resources, and cloud-native frameworks with high-performance systems for efficient workflows. How-

ever, prevalent container networking architectures, reliant on overlay networks, often incur performance overhead, particularly affecting co-processes on the same node. To address this challenge, [Njavro et al. [Njavro et al. 2022] explore leveraging the Nvidia Bluefield 2 SmartNIC to offload Docker overlay networks, aiming to enhance support for cloud-native overlay networks alongside existing HPC workloads sensitive to system noise. Through characterizing different offloading approaches, investigating the feasibility of DPU offload, and evaluating performance benefits, the paper contributes to optimizing the integration of cloud-native technologies with traditional HPC environments, enabling efficient coexistence and operation of diverse workloads in hybrid computing infrastructures.

2.6.3. ASIC-based SmartNICs

Deploying machine learning algorithms in high-throughput networking environments poses significant challenges, with existing approaches often focusing on offline post-processing or introducing high latency. To address this, Xavier et al. [Xavier et al. 2021] introduce a framework to create simple yet accurate machine-learning models that are deployable directly into the data plane with acceptable performance degradation. Their approach involves translating these models, tailored for individual packets or flows, into the P4 language, a crucial step towards in-network deployment. Through validation using an intrusion detection use case and deployment on a Netronome SmartNIC (Agilio CX 2x10GbE), they demonstrate the feasibility of achieving high accuracy (above 95%) with minimal performance impact, even with many flows. This work signifies a step forward in integrating machine learning into network devices, leveraging the capabilities of programmable switches and SmartNICs to enhance real-world networking applications.

Integrating ML models into programmable networking devices has sparked interest in leveraging data plane capabilities for autonomous network management. However, challenges persist in adapting unsupervised ML algorithms to the constraints of programmable forwarding devices. In response, Cannarozzo et al [Cannarozzo et al. 2024] proposes SPINNER, an innovative approach for in-network flow clustering directly within the data plane. SPINNER maps network flows to multidimensional vectors and dynamically assigns them to clusters as packets traverse the programmable device. By clustering flows in the data plane, SPINNER enables line-rate flow balancing, congestion control, and differentiated services. Implemented in the SmartNIC Netronome NFP 4000, SPINNER demonstrates promising results, enhancing TCP throughput by up to 2X compared to vanilla TCP with minimal incurred latency.

The TCP protocol is the backbone of modern data networking, ensuring reliable data transfer between endpoints. Yet, its adherence to protocol standards often incurs performance penalties, particularly in short-lived connections and layer-7 proxying scenarios. Addressing this challenge, Moon et al. [Moon et al. 2020] leverages Netronome Agilio LX 40GbE to accelerate TCP processing, presenting a dual-stack design that offloads select operations to the NIC stack while maintaining control-plane functions at the host stack. This approach significantly reduces CPU and memory bandwidth overhead on the host stack, allowing applications to focus on core functionality. Moreover, Moon et al. [Moon et al. 2020] per-flow offloading decision offers flexibility, ensuring optimal performance under varying conditions. Despite challenges in maintaining consistency across

host and NIC stacks, the work effectively manages complexity, achieving notable performance gains. Evaluation results demonstrate its superiority over existing TCP stacks and its substantial impact on real-world applications like Redis and HAProxy, making it a significant contribution to network performance optimization.

2.7. Diving into SmartNICs: Hands-on experiences

In this section, we are going to deep dive into practical hands-on examples using current available SmartNICs. Note that all used coding and additional scripting materials are available at our public repository.[10]

2.7.1. Hands-on with Nvidia BlueField

In this example, we will demonstrate the step-by-step process of setting up a DOCA application. Firstly, we need to set up our environment correctly. Afterward, we will describe a simple DOCA Flow application that utilizes DOCA pipes to offload packet processing to the BlueField SmartNIC. It is important to note that our settings can also be replicated in available testbeds such as FABRIC.[11].

2.7.1.1. Setting up the SmartNIC

First, we need to install the DOCA framework. This is done on the host side using Ubuntu 20.04 (Kernel 5.15-0-67-generic). To install DOCA utils, download the file and follow the installation instructions.

```
smartness@host# dpkg -i doca-host-repo-ubuntu2204*_amd64.deb
smartness@host# apt-get update
smartness@host# apt install doca-runtime
smartness@host# apt install doca-tools
smartness@host# apt install -y doca-extra
smartness@host# apt install pv
```

Then, we need to initialize the Mellanox Software Tools service (a.k.a. MST). The MST command is used to create special files that represent Mellanox devices in the directory /dev/mst. This command loads appropriate kernel modules and saves PCI configuration headers in the temporary directory. Once this command is completed successfully, the MST driver is ready to work, and other Mellanox tools can be invoked.

```
smartness@host# mst start
smartness@host# mst status -v
```

The following is an example of the output you should expect to see when running the command `mst status -v`. The output will contain the PCI address (e.g. 03:00.0), the MST address (e.g. `mt41686_pciconf0`), and the name of the network interface that is exposed to the operating system (e.g. `net-enp3s0f0`).

[10] `https://github.com/smartness2030/sbrc24-minicurso-smartnic`
[11] `https://fabricmc.net/`

```
MST modules:
------------
    MST PCI module is not loaded
    MST PCI configuration module loaded
PCI devices:
------------
DEVICE_TYPE             MST                                   PCI       RDMA      NET
BlueField2(rev:1)       /dev/mst/mt41686_pciconf0.1           03:00.1   mlx5_1    net-enp3s0f1

BlueField2(rev:1)       /dev/mst/mt41686_pciconf0             03:00.0   mlx5_0    net-enp3s0f0
```

To get started, we must first initialize the RSHIM service. The RShim serves as the current interface for managing the SoC (System on a Chip). It enables an external agent, such as the host CPU or BMC, to operate the DPU (Data Processing Unit) and monitor its operational state. With this interface, the DPU can be provisioned, Arm cores can be reset, and logs can be obtained. On the host, the SoC management interface driver exposes a virtual Ethernet device called `tmfifo_net0`. This virtual Ethernet device functions as a peer-to-peer tunnel that links the host and the DPU OS. In response, the DPU OS sets up a comparable device. The BFB (Bootloader Firmware Binary) images within the DPU OS are tailored to set up the DPU end of this connection, assigning a predefined IP address of `192.168.100.2/30` within the DPU OS.

```
smartness@host# systemctl enable rshim
smartness@host# systemctl start rshim
smartness@host# systemctl status rshim
```

The following is the expected output of the RSHIM initialization mentioned above:

```
rshim.service - rshim driver for BlueField SoC
  Loaded: loaded (/lib/systemd/system/rshim.service; enabled; vendor preset: enabled)
  Active: active (running) since Wed 2024-04-03 13:19:25 -03; 4 days ago
         Docs: man:rshim(8)
  Main PID: 1322 (rshim)
     Tasks: 6 (limit: 18833)
     Memory: 1.1M
     CGroup: /system.slice/rshim.service
             1322 /usr/sbin/rshim

abr 03 13:19:25 Hydra-202 systemd[1]: Starting rshim driver for BlueField SoC...
abr 03 13:19:25 Hydra-202 systemd[1]: Started rshim driver for BlueField SoC.
abr 03 13:19:25 Hydra-202 rshim[1322]: Probing pcie-0000:03:00.2
abr 03 13:19:25 Hydra-202 rshim[1322]: create rshim pcie-0000:03:00.2
abr 03 13:19:26 Hydra-202 rshim[1322]: rshim0 attached
```

If you are setting up the SmartNIC for the first time, you may need to install or reinstall the operating system in the SoC SmartNIC. To get started, you will need to download the Linux image and then use NVIDIA toolchain to install it. In this tutorial, we will be using Ubuntu `22.04`, which is provided by NVIDIA.

```
smartness@host# wget https://content.mellanox.com/BlueField/
                BFBs/Ubuntu22.04/DOCA_2.0.2_BSP_4.0.3_
                Ubuntu_22.04-10.23-04.prod.bfb
```

We need to set a configuration file and install the Ubuntu image with a predefined password using the `bfb-install` tool. The following steps will help you perform these actions.

```
smartness@host# hash=`openssl passwd -1`
smartness@host# echo ubuntu_PASSWORD='$hash' >> bf.cfg

smartness@host# bfb-install --rshim /dev/rshim0
                            --bfb <image_path.bfb>
                            --config bf.cfg
```

The operating system has been installed successfully and is now ready to be used. However, before using it, we should change the SmartNIC operation mode. For the purpose of this tutorial, we will be using the SmartNIC in the DPU mode. The NVIDIA BlueField offers three modes of operation: DPU mode (or embedded function ECPF), zero-trust mode, and NIC mode. In DPU mode, the NIC resources and functionalities are owned by the embedded Arm subsystem. All network communication destined for the host passes through a virtual switch control plane hosted on the Arm cores, before reaching the host. In this operational mode, the DPU acts as the trusted entity, overseen by both data center and host administrators. Its responsibilities include loading network drivers, resetting interfaces, toggling interface states, firmware updates, and altering the operational mode of the DPU device. The following steps will help you change the DPU operation mode.

```
smartness@host# #mst status -v
#Use the above command to check the MST device addrress

smartness@host# mlxconfig -d /dev/mst/mt41686_pciconf0
                s INTERNAL_CPU_MODEL=1
smartness@host# mlxconfig -d /dev/mst/mt41686_pciconf0.1
                s INTERNAL_CPU_MODEL=1
smartness@host# reboot
```

In order to access the SoC subsystem, we must configure a valid IP address for the virtual interface called `tmfifo_net0`. The SoC management interface has a peer-to-peer tunnel that connects the host and the DPU OS, and it is automatically set up in the `192.168.100.0/30` subnet. For this tutorial, use the following command to assign `192.168.100.2` to the `tmfifo_net0` interface.

```
smartness@host# ip address add 192.168.100.2/30 dev tmfifo_net0
```

Also, it is necessary to configure NAT rules on the host side to enable the OS in NIC to access the Internet. The following steps will configure `iptables` on the host

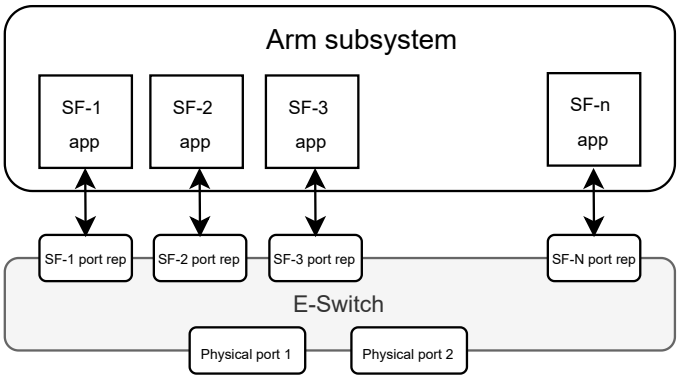


Figure 2.14. Scalable Function in the Nvidia BlueField SoC. Adapted from NVIDIA DOCA documentation [NVIDIA 2024].

side.

```
#replace with the interface connected to the Internet
smartness@host# OUTFACE=enp6s0
smartness@host# NIC=tmfifo_net0
smartness@host# echo 1 > /proc/sys/net/ipv4/ip_forward
smartness@host# iptables -A FORWARD -i $NIC -o $OUTFACE \
                         -j ACCEPT
smartness@host# iptables -A FORWARD -i $OUTFACE -o $NIC \
                         -m state --state ESTABLISHED,RELATED \
                         -j ACCEPT
smartness@host# iptables -t nat -A POSTROUTING -o $OUTFACE \
                         -j MASQUERADE
```

2.7.1.2. Accessing the SoC and Preliminary Configurations

After configuring the host side, we can now access the SoC subsystem. To access the SmartNIC SoC operating system, we can use `ssh` as follows.

```
smartness@host# ssh ubuntu@192.168.100.2
```

If everything goes smoothly, you will be logged into the SmartNIC operating system. By default, the SmartNIC OS comes with an Open vSwitch (OVS) that has two default bridges named `ovsbr1` and `ovsbr2`. In each bridge, there is a physical port attached, namely `p1` and `p2`. Each OVS bridge has at least one host interface representator, such as `pf1hpf` and `pf0hpf`. These interface representators are used to connect with the host interfaces (e.g., `net-enp3s0f1`). Lastly, each OVS bridge comes with a Scalable Function (e.g., `en3f1pf1sf0` and `en3f0pf0sf0`). Scalable Functions (SFs), or sub-functions, are similar to Virtual Functions (VFs) that are part of a Single Root I/O Virtualization (SR-IOV) solution. An SF is a lightweight function that has a parent PCIe function on which it is deployed. The SF, therefore, has access to the capabilities and resources of its parent PCIe function, and it has its own function capabilities and resources. This means that an SF also has its own dedicated queues (i.e., txq, rxq). Figure 2.14

illustrates the relationship between internal ports and SF representators.

```
root@localhost:/home/ubuntu# ovs-vsctl show
e350e1eb-a1f2-4eb8-b9ad-8bb2a3a7f86a
    Bridge ovsbr2
        Port p1
            Interface p1
        Port pf1hpf
            Interface pf1hpf
        Port en3f1pf1sf0
            Interface en3f1pf1sf0
        Port ovsbr2
            Interface ovsbr2
                type: internal
    Bridge ovsbr1
        Port pf0hpf
            Interface pf0hpf
        Port ovsbr1
            Interface ovsbr1
                type: internal
        Port p0
            Interface p0
        Port en3f0pf0sf0
            Interface en3f0pf0sf0
    ovs_version: "2.17.7-e054917"
```

In Figure 2.14, you can see how physical ports interact with SFs. SFs are used by DOCA applications and communicate with the physical port by SF representators. It is possible to dynamically add, remove, or modify SFs according to the application's needs using the `mlxdevm port` command. Next, we can use the `mlxdevm port show` command to list all registered SFs and their features. For instance, we can see the SF identifier (e.g., 229408) and to which physical port each SF is attached. For example, the SF `pci/0000:03:00.0/229408` is attached to physical port 0.

```
root@smartNIC:/# /opt/mellanox/iproute2/sbin/mlxdevm port show
```

```
pci/0000:03:00.0/229408: type eth netdev en3f0pf0sf0 flavour pcisf controller 0
                                                              pfnum 0 sfnum 0
 function:
 hw_addr 02:eb:3d:64:53:86 state active opstate attached roce true max_uc_macs 128
                                                              trust off

pci/0000:03:00.1/294944: type eth netdev en3f1pf1sf0 flavour pcisf controller 0
                                                              pfnum 1 sfnum 0
 function:
 hw_addr 02:43:b8:2f:7a:fb state active opstate attached roce true max_uc_macs 128
                                                              trust off
```

Next, we will create, configure and deploy two Scalable Functions. For more information, please refer to DOCA Scalable Function documentation[12].

[12] `https://docs.nvidia.com/doca/archive/doca-v2.2.0/scalable-functions/index.html`

```
root@smartNIC:/# #Command syntax
                 #/opt/mellanox/iproute2/sbin/mlxdevm port \
                 function set pci/<pci_address>/<sf_index>  \
                 hw_addr <MAC address> trust on state active

root@smartNIC:/# /opt/mellanox/iproute2/sbin/mlxdevm port \
                 function set pci/0000:03:00.0/229409  \
                 hw_addr 00:00:00:00:04:0 trust on state active

root@smartNIC:/# /opt/mellanox/iproute2/sbin/mlxdevm port \
                 function set pci/0000:03:00.0/294944  \
                 hw_addr 00:00:00:00:05:0 trust on state active
```

We then verify if the just created SFs are correctly deployed using `devlink dev show` command.

```
root@smartNIC:/# devlink dev show
```

Last, we need to connect the SFs' interfaces to the OVS bridges. This will enable communication between SFs, physical interfaces, and the host interface representators.

```
root@smartNIC:/# ovs-vsctl add-port ovsbr1 en3f1pf1sf0
root@smartNIC:/# ovs-vsctl add-port ovsbr2 en3f1pf1sf1
root@smartNIC:/# ovs-vsctl show
```

In addition to that, we need to add flow entries to OVS (or to the eSwitch in the BlueField) so that packets can move between the physical interface and SFs.

```
root@smartNIC:/# sudo ovs-ofctl add-flow ovsbr1 in_port=p0,\
                 actions=output:en3f0pf0sf1
root@smartNIC:/# sudo ovs-ofctl add-flow ovsbr2 \
                 in_port=en3f1pf1sf1,actions=output:p1
root@smartNIC:/# sudo ovs-ofctl add-flow ovsbr2 in_port=p1,\
                 actions=output:en3f1pf1sf1
root@smartNIC:/# sudo ovs-ofctl add-flow ovsbr1 \
                 in_port=en3f0pf0sf1,actions=output:p0
root@smartNIC:/# sudo ovs-ofctl dump-flows ovsbr1
root@smartNIC:/# sudo ovs-ofctl dump-flows ovsbr2
```

2.7.1.3. Programming the SmartNIC with DOCA

We will now go over a simple DOCA application that is based on the DOCA Flow library. Our environment is now set up for the DOCA application. Our objective is to gain a high-level understanding of the code so that we can deploy and debug it with ease. As mentioned in Section 2.4, a DOCA Flow application is made up of pipes, where each pipe can have matching and actions. We will start by discussing the main building block of the sample code. You can find the complete code at our public repository[13].

[13] `https://github.com/smartness2030/sbrc24-minicurso-smartnic`

The first building block consists of a function that creates a DOCA Flow pipe. The function creates a simple pipe that matches on IPv4 (`src_ip`, `dst_ip`, `src_port`, `dst_port`). The function `create_hairpin_pipe` creates a hairpin pipe, which allows traffic to ingress and egress through the same physical port. It initializes various data structures such as `struct doca_flow_match`, `struct doca_flow_actions`, and `struct doca_flow_fwd`, and configures the pipe settings in `struct doca_flow_pipe_cfg`. Matching criteria are established for IPv4/TCP traffic with wildcard IP addresses and port numbers. Forwarding rules are set to direct traffic to another port. Finally, the function calls `doca_flow_pipe_create` to create the hairpin pipe based on the configured settings and returns the result.

```
static doca_error_t
create_hairpin_pipe(struct doca_flow_port *port, int port_id,
                    struct doca_flow_pipe **pipe)
{
        struct doca_flow_match match;
        struct doca_flow_actions actions, *actions_arr[NB_ACTIONS_ARR];
        struct doca_flow_fwd fwd;
        struct doca_flow_pipe_cfg pipe_cfg;

        memset(&match, 0, sizeof(match));
        memset(&actions, 0, sizeof(actions));
        memset(&fwd, 0, sizeof(fwd));
        memset(&pipe_cfg, 0, sizeof(pipe_cfg));

        pipe_cfg.attr.name = "HAIRPIN_PIPE";
        pipe_cfg.attr.type = DOCA_FLOW_PIPE_BASIC;
        pipe_cfg.match = &match;
        actions_arr[0] = &actions;
        pipe_cfg.actions = actions_arr;
        pipe_cfg.attr.is_root = true;
        pipe_cfg.attr.nb_actions = NB_ACTIONS_ARR;
        pipe_cfg.port = port;

        /* 5 tuple match */
        match.outer.l4_type_ext = DOCA_FLOW_L4_TYPE_EXT_TCP;
        match.outer.l3_type = DOCA_FLOW_L3_TYPE_IP4;
        match.outer.ip4.src_ip = 0xffffffff;
        match.outer.ip4.dst_ip = 0xffffffff;
        match.outer.tcp.l4_port.src_port = 0xffff;
        match.outer.tcp.l4_port.dst_port = 0xffff;

        /* forwarding traffic to other port */
        fwd.type = DOCA_FLOW_FWD_PORT;
        fwd.port_id = port_id ^ 1;

        return doca_flow_pipe_create(&pipe_cfg, &fwd, NULL, pipe);
}
```

Another important building block of of sample application is the function that adds pipe entries. Similarly to P4 (or OpenFlow), the function adds entries that are going to be used to match against the information of incoming packets. In the following example, we add a single entry having `dst_ip` = `8.8.8.8`, `src_ip` = `1.2.3.4`, `dst_port` = `80`, and `src_port` = `1234`. The function `add_hairpin_pipe_entry` is used to add an entry to a hairpin pipe in a networking context. It takes a `struct doca_flow_pipe` representing the pipe and a `struct doca_flow_port` representing the port as parameters. Inside the function, it initializes several data structures including `struct`

`doca_flow_match` and `struct doca_flow_actions`. It sets up matching criteria for IPv4/TCP traffic with specific source and destination IP addresses and port numbers. The function then adds this entry to the pipe using `doca_flow_pipe_add_entry`. Subsequently, it processes the added entry on the specified port using `doca_flow_entries_process` and checks the processing status. This function processes entries in the queue. The application must invoke this function to complete flow rule offloading and to receive the flow rule's operation status. If the entry is successfully processed, it returns `DOCA_SUCCESS`, otherwise, it returns an appropriate error code.

```
static doca_error_t
add_hairpin_pipe_entry(struct doca_flow_pipe *pipe, struct doca_flow_port *port)
{
        struct doca_flow_match match;
        struct doca_flow_actions actions;
        struct doca_flow_pipe_entry *entry;
        struct entries_status status;
        doca_error_t result;
        int num_of_entries = 1;

        /* example 5-tuple to forward */
        doca_be32_t dst_ip_addr = BE_IPV4_ADDR(8, 8, 8, 8);
        doca_be32_t src_ip_addr = BE_IPV4_ADDR(1, 2, 3, 4);
        doca_be16_t dst_port = rte_cpu_to_be_16(80);
        doca_be16_t src_port = rte_cpu_to_be_16(1234);

        memset(&status, 0, sizeof(status));
        memset(&match, 0, sizeof(match));
        memset(&actions, 0, sizeof(actions));

        match.outer.ip4.dst_ip = dst_ip_addr;
        match.outer.ip4.src_ip = src_ip_addr;
        match.outer.tcp.l4_port.dst_port = dst_port;
        match.outer.tcp.l4_port.src_port = src_port;

        result = doca_flow_pipe_add_entry(0, pipe, &match, &actions, NULL, NULL,
                0, &status, &entry);
        if (result != DOCA_SUCCESS)
                return result;

        result = doca_flow_entries_process(port, 0, DEFAULT_TIMEOUT_US
                                           , num_of_entries);
        if (result != DOCA_SUCCESS)
                return result;

        if (status.nb_processed != num_of_entries || status.failure)
                return DOCA_ERROR_BAD_STATE;

        return DOCA_SUCCESS;
}
```

Then, we initialize the DOCA Flow API function and call the previously defined functions to create and populate DOCA pipes. In this example, we create a pipe for each existing port. The `flow_hairpin` function orchestrates the configuration of hairpin networking scenarios using the DOCA library. It initializes the necessary DOCA flow resources and ports, then enters a loop where it iterates through each port, creating a hairpin pipe and adding an entry to it. After each iteration, it waits for a brief period for packets to arrive. This process continues indefinitely until manually stopped. Upon completion, it properly stops the DOCA flow ports and releases the associated resources.

```
#include <string.h>
#include <unistd.h>
#include <rte_byteorder.h>
#include <doca_log.h>
#include <doca_flow.h>
#include "flow_common.h"

DOCA_LOG_REGISTER(FLOW_HAIRPIN);

doca_error_t
flow_hairpin(int nb_queues)
{
        int inc = 0;
        int nb_ports = 2;
        struct doca_flow_resources resource = {0};
        uint32_t nr_shared_resources[DOCA_FLOW_SHARED_RESOURCE_MAX] = {0};
        struct doca_flow_port *ports[nb_ports];
        struct doca_flow_pipe *pipe;
        doca_error_t result;
        int port_id;

        result = init_doca_flow(nb_queues, "vnf,hws", resource, nr_shared_resources);
        if (result != DOCA_SUCCESS) {
                DOCA_LOG_ERR("Failed to init DOCA Flow: %s",
                             doca_get_error_string(result));
                return result;
        }

        result = init_doca_flow_ports(nb_ports, ports, true);
        if (result != DOCA_SUCCESS) {
                DOCA_LOG_ERR("Failed to init DOCA ports: %s",
                             doca_get_error_string(result));
                doca_flow_destroy();
                return result;
        }

        while(1){
        for (port_id = 0; port_id < nb_ports; port_id++) {
                result = create_hairpin_pipe(ports[port_id], port_id, &pipe);
                if (result != DOCA_SUCCESS) {
                        DOCA_LOG_ERR("Failed to create pipe: %s",
                                     doca_get_error_string(result));
                        stop_doca_flow_ports(nb_ports, ports);
                        doca_flow_destroy();
                        return result;
                }

                result = add_hairpin_pipe_entry(pipe, ports[port_id]);
                if (result != DOCA_SUCCESS) {
                        DOCA_LOG_ERR("Failed to add entry: %s",
                                      doca_get_error_string(result));
                        stop_doca_flow_ports(nb_ports, ports);
                        doca_flow_destroy();
                        return result;
                }
        }

        DOCA_LOG_INFO("tst2: %i", inc);
        DOCA_LOG_INFO("Wait few seconds for packets to arrive");
        sleep(5);
        }

        stop_doca_flow_ports(nb_ports, ports);
        doca_flow_destroy();
        return DOCA_SUCCESS;
}
```

Overall, the function provides a streamlined approach to setting up and managing hairpin configurations across multiple ports for networking tasks. The full code is available at our GitHub.[14] Next, we compile and run the code. To compile, we must ensure that DOCA is on the Linux path before running the code.

```
root@smartNIC:/# export PKG_CONFIG_PATH=:/opt/mellanox/doca/ \
                 lib/aarch64-linux gnu/pkgconfig:/opt/ \
                 mellanox/dpdk/lib/aarch64-linux-gnu/pkgconfig \
                 \:/opt/mellanox/flexio/lib/pkgconfig
```

Then, we compile with Meson and execute the DOCA application.

```
root@smartNIC:/app # meson build
root@smartNIC:/app # cd build
root@smartNIC:/app/build # ninja
root@smartNIC:/app/build # ./doca_flow_hairpin \
           -a auxiliary:mlx5_core.sf.4,dv_flow_en=2 \
           -a auxiliary:mlx5_core.sf.5,dv_flow_en=2 -- -l 60
```

Once we have done that, we can start sending packets to our DOCA application. Figure 2.15(a) illustrates the TRex Traffic Generator[15] sending TCP packets to the BlueField-2 Soc. Observe that the BlueField-2 is handling almost 30Mpps (millions of packet per second) in both directions. As the whole packet processing is offloaded using the DOCA application discussed, we observe that the ARM cores in the SoC subsystem are completely not used (see Figure 2.15(b).

2.7.2. Hands-on with Nvidia Connectx SmartNIC offload capabilities

In this example, we will showcase the offload capabilities of the Nvidia Connectx SmartNIC. We will compare the performance of a software router with and without offloads. To implement the software router, we will use Vector Packet Processing (VPP) [Barach et al. 2018], a high-performance network stack that supports different data planes (Linux, RDMA, and DPDK). VPP can be used as vSwitches, vRouters, Gateways, Firewalls, and Load-Balancers. The reference traffic generator from DPDK, Pktgen-DPDK, will be used to stress test VPP. It can generate 100Gbps using a single CPU core. We will conduct the tests using Nvidia Connectx-6 SmartNIC, which is available on testbeds like FABRIC and RNP Testbed Service[16] to facilitate replication of this example.

2.7.2.1. Setting up the VPP SmartNIC

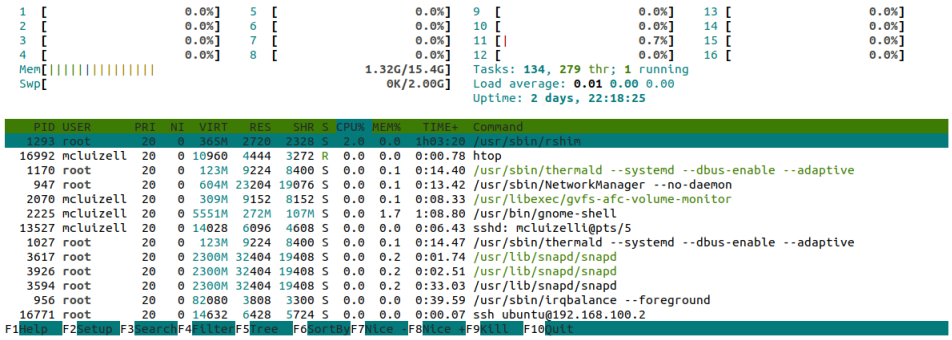
To begin with, you need to recognize the ConnectX SmartNIC network interfaces and take note of their name, PCI address, and MAC address. To list all network interfaces, including the necessary details, you can use the following show command. However, only rows 3 and 4 will be utilized, which correspond to ConnectX-6 physical interfaces.

[14] https://github.com/smartness2030/sbrc24-minicurso-smartnic
[15] https://trex-tgn.cisco.com/
[16] https://www.rnp.br/en/research-development/testbeds

owner	mcluizelli	mcluizelli	
link	UP	UP	
state	TRANSMITTING	TRANSMITTING	
speed	25 Gb/s	25 Gb/s	
CPU util.	20.18%	20.18%	
--			
Tx bps L2	15.03 Gbps	14.98 Gbps	30.01 Gbps
Tx bps L1	19.73 Gbps	19.66 Gbps	39.38 Gbps
Tx pps	29.35 Mpps	29.25 Mpps	58.6 Mpps
Line Util.	78.9 %	78.63 %	

Rx bps	15.03 Gbps	14.98 Gbps	30.01 Gbps
Rx pps	29.35 Mpps	29.25 Mpps	58.6 Mpps
----			
opackets	368752720	368998365	737751085
ipackets	366671412	366796726	733468138
obytes	23600184816	23615908594	47216093410
ibytes	23466971520	23474992529	46941964049
tx-pkts	368.75 Mpkts	369 Mpkts	737.75 Mpkts
rx-pkts	366.67 Mpkts	366.8 Mpkts	733.47 Mpkts
tx-bytes	23.6 GB	23.62 GB	47.22 GB
rx-bytes	23.47 GB	23.47 GB	46.94 GB
-----			
oerrors	0	0	0
ierrors	0	0	0

(a) TRex Traffic Generator when running sample application on BlueField-2.



(b) BlueField-2 SoC CPU usage using `htop` command in Linux.

Figure 2.15. Running sample application on BlueField-2 SoC.

```
ubuntu@vpp-node:~$ sudo lshw -class network -businfo
```

```
Bus info          Device      Class          Description
========================================================
pci@0000:03:00.0              network        Virtio network device
virtio@1          enp3s0      network        Ethernet interface
pci@0000:07:00.0  enp7s0np0    network        MT28908 Family [ConnectX-6]
pci@0000:08:00.0  enp8s0np0    network        MT28908 Family [ConnectX-6]
pci@0000:09:00.0  enp9s0      network        MT28908 Family
```

The first column of the output displays the PCI address, which is used for both `VPP_IF#_PCI` and `VPP_IF#_NAME`. `VPP_IF#_NAME` is used as the interface name on VPP when using the DPDK backend. The second column represents the interface name on Linux, which is denoted by `VPP_IF#`.

```
ubuntu@vpp-node:~$ VPP_IF1_PCI=07:00.0
ubuntu@vpp-node:~$ VPP_IF1_NAME="HundredGigabitEthernet7/0/0"
ubuntu@vpp-node:~$ VPP_IF1=enp7s0np0
ubuntu@vpp-node:~$ VPP_IF2=enp8s0np0
ubuntu@vpp-node:~$ VPP_IF2_NAME="HundredGigabitEthernet8/0/0"
ubuntu@vpp-node:~$ VPP_IF2_PCI=08:00.0
```

As an initial baseline, we will run VPP without any offload, using only the Linux networking stack. To increase performance, we need to configure the IP address on both interfaces and set the MTU to 9000.

```
ubuntu@vpp-node:~$ sudo ifconfig $VPP_IF1 mtu 9000
ubuntu@vpp-node:~$ sudo ifconfig $VPP_IF1 192.168.0.2/24
ubuntu@vpp-node:~$ sudo ifconfig $VPP_IF2 mtu 9000
ubuntu@vpp-node:~$ sudo ifconfig $VPP_IF2 192.168.1.2/24
```

Enabling IP Forwarding is necessary to route packets through different interfaces. If using a firewall on your Linux host, consider adding the following rules using `nft` or `iptables`.

```
ubuntu@vpp-node:~$ sudo sysctl -w net.ipv4.ip_forward=1
```

```
# nft rules
ubuntu@vpp-node:~$ sudo nft insert rule ip filter \
                   FORWARD iifname $VPP_IF1 oifname $VPP_IF2 \
                    counter accept
ubuntu@vpp-node:~$ sudo nft insert rule ip filter \
                   FORWARD iifname $VPP_IF2 oifname $VPP_IF1 \
                    counter accept

# iptables rules
ubuntu@vpp-node:~$ sudo iptables -A FORWARD -i $VPP_IF1 \
                   -o $VPP_IF2 -j ACCEPT
ubuntu@vpp-node:~$ sudo iptables -A FORWARD -i $VPP_IF2 \
                   -o $VPP_IF1 -j ACCEPT
```

For this example, we will be using Version 24.02 of VPP. To deploy and run it, you can use the docker compose file provided in the `vpp` folder of the GitHub repository. Initially, VPP will run with a custom configuration file called `startup-nodpdk.conf` which disables DPDK and enables the use of other datapaths.

```
...
plugins {
        ## Enable all plugins by default and then selectively disable specific plugins
        plugin dpdk_plugin.so { disable }
}
...
```

Start VPP container in the background using docker compose up with “-d” flag

```
ubuntu@vpp-node:~$ docker compose up -d
```

To simplify running `vppctl` from within the container, create an alias using the command line. This tool is used to configure VPP, including the interfaces and IP addresses, based on the variables set earlier in this section. If you want to use native Linux interfaces on VPP without any additional offload, you can specify the `host-interface` type.

```
ubuntu@vpp-node:~$ alias vppctl="docker compose exec vpp vppctl"
ubuntu@vpp-node:~$ vppctl create host-interface name $VPP_IF1
ubuntu@vpp-node:~$ vppctl set int ip address host-$VPP_IF1 \
                   192.168.0.2/24
ubuntu@vpp-node:~$ vppctl set interface state host-$VPP_IF1 up
ubuntu@vpp-node:~$ vppctl create host-interface name $VPP_IF2
ubuntu@vpp-node:~$ vppctl set int ip address host-$VPP_IF2 \
                   192.168.1.2/24
ubuntu@vpp-node:~$ vppctl set interface state host-$VPP_IF2 up
```

Use the following command to confirm that the interface name, IP address, and state are as intended.

```
ubuntu@vpp-node:~$ vppctl show interface address
```

```
host-enp7s0np0 (up):
  L3 192.168.0.2/24
host-enp8s0np0 (up):
  L3 192.168.1.2/24
local0 (dn):
```

In the following section, `VPP_IF1 MAC` address needs to be configured on Pktgen-DPDK, so take note of it using the `ip link` command and referring to the value just after `link/ether`, which in the following example is `10:70:fd:e5:cd:60`.

```
ubuntu@vpp-node:~$ ip link show $VPP_IF1
```

```
3: enp7s0np0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP mode \
              DEFAULT group default qlen 1000
    link/ether 10:70:fd:e5:cd:60 brd ff:ff:ff:ff:ff:ff
```

2.7.2.2. Setting up the Pktgen-DPDK SmartNIC

DPDK applications use large blocks of contiguous memory called `hugepages`. These blocks can be either 2MB or 1GB in size. For instance, to run Pktgen-DPDK, we need to

allocate 2048 blocks of 2MB in size.

```
ubuntu@pktgen-node:~$ echo 2048 | sudo tee /sys/kernel/mm/hugepages/ \
                      hugepages-2048kB/nr_hugepages
```

Next, we have to create a container that has Pktgen-DPDK 24.03.1 and DPDK 24.03 by utilizing the Dockerfile in the `pktgen-dpdk` folder of the GitHub repository. We can leverage Docker Compose to build the Pktgen-DPDK container.

```
ubuntu@pktgen-node:~$ docker compose build
```

Pktgen-DPDK is an interactive application. To enable it to refresh the screen and display real-time information, we use the `docker-compose run` command. We also need to provide Pktgen-DPDK with the PCI address of the interfaces and the CPU cores that we intend to use.

```
ubuntu@pktgen-node:~$ docker compose run --rm pktgen-dpdk \
                      pktgen -a 08:00.0 -a 09:00.0 -l 0-4 \
                      -n 3 -- -m "[1:3].0, [2:4].1" -j
```

Afterward, Pktgen-DPDK will provide a prompt where we can set the parameters of the traffic we want to generate. In this example, we will set the protocol to UDP, the packet size to 9000, and the destination MAC address to `VPP_IF1` MAC address noted at the end of section 2.8.2.1. The command `start 0` initiates the packet generation on port 0 using the provided profile. Additionally, `enable 1 process` will allow port 1 to reply to ARP requests, which are necessary to establish end-to-end communication.

```
Pktgen:/> set 0 proto udp
Pktgen:/> set 0 size 9000
Pktgen:/> set 0 dst mac 10:70:fd:e5:cd:60
Pktgen:/> start 0
Pktgen:/> enable 1 process
```

Pktgen-DPDK will start updating the statistics on the top of the page, where we can track the transmitted rate (TX) on port 0 and the receive rate on port 1 (RX) on line 6 (MBits/s Rx/Tx). In the following example, Pktgen-DPDK is transmitting 98.4 Gbps (100753 Mbps / 1024) and receiving only 17 Gbps (17468 Mbps / 1024) using the Linux datapath.

```
Ports 0-1 of 2   <Main Page>  Copyright(c) <2010-2023>, Intel Corporation
  Port:Flags        : 0:P------      Single 1:P--I---      Single
Link State          :       <UP-100000-FD>       <UP-100000-FD>
---Total Rate---
Pkts/s Rx           :                    0               242191
242191
       Tx           :              1396864                    0
1396864
MBits/s Rx/Tx       :              0/100753              17468/0
17468/100753
...
```

It is important to keep Pktgen running because we will use it to verify the throughput of the SmartNIC using offloads with RDMA and DPDK.

2.7.2.3. Setting up the VPP SmartNIC with RDMA offload

VPP provides a driver for RDMA (Remote Direct Memory Access) devices[17], which uses RDMA APIs (Application Programming Interfaces) that are available on Nvidia ConnectX SmartNICS. This driver helps to offload Ethernet packets. To use it, you need to restart the VPP container to clean up the configurations and configure the RDMA interfaces.

```
docker compose restart
```

In this example, RDMA type interfaces are created to use the RDMA device driver. Physical interfaces are referenced using the `VPP_IF#` variable defined earlier. You can use the following command to confirm that the interface name, IP address, and state are as intended.

```
ubuntu@vpp-node:~$ alias vppctl="docker compose exec vpp vppctl"
ubuntu@vpp-node:~$ vppctl create interface rdma host-if $VPP_IF1 name rdma-0
ubuntu@vpp-node:~$ vppctl set int ip address rdma-0 192.168.0.2/24
ubuntu@vpp-node:~$ vppctl set interface state rdma-0 up
ubuntu@vpp-node:~$ vppctl create interface rdma host-if $VPP_IF2 name rdma-1
ubuntu@vpp-node:~$ vppctl set int ip address rdma-1 192.168.1.2/24
ubuntu@vpp-node:~$ vppctl set interface state rdma-1 up
```

Again, use the following command to confirm that the interface name, IP address, and state are as intended.

```
ubuntu@vpp-node:~$ vppctl show interface address
```

[17] `https://fd.io/docs/vpp/master/developer/devicedrivers/rdma.html`

```
local0 (dn):
rdma-0 (up):
  L3 192.168.0.2/24
rdma-1 (up):
  L3 192.168.1.2/24
```

Once you have confirmed that everything is correctly set up, go back to Pktgen-DPDK and verify that the receiving rate has increased by a factor of approximately 2.6 times to 44 Gbps (45348 Mbps / 1024).

```
| Ports 0-1 of 2   <Main Page>  Copyright(c) <2010-2023>, Intel Corporation
  Port:Flags        : 0:P------      Single 1:P--I---      Single
Link State          :       <UP-100000-FD>       <UP-100000-FD>
---Total Rate---
Pkts/s Rx           :                   0              628729
628729
       Tx           :              1385728                   0
1385728
MBits/s Rx/Tx       :              0/99949             45348/0
45348/99949
```

2.7.2.4. Setting up the VPP SmartNIC with DPDK offload

In this section, we will be using DPDK to accelerate VPP. As previously mentioned, we need to allocate hugepages to use DPDK.

```
ubuntu@vpp-node:~$ echo 2048 |
                   sudo tee /proc/sys/vm/nr_hugepages
```

To begin with, stop the VPP container and run it by explicitly setting the `compose-yaml` file using "-f". This will enable VPP's default configuration file and initialize it with the DPDK plugin.

```
ubuntu@vpp-node:~$ docker compose down
ubuntu@vpp-node:~$ docker compose -f compose.yaml up -d
```

VPP will automatically detect and create interfaces that are compatible with the DPDK plugin. You can verify this by using the following command.

```
ubuntu@vpp-node:~$ alias vppctl="docker compose exec vpp vppctl"
ubuntu@vpp-node:~$ vppctl show interface
```

```
              Name                Idx   State          MTU
HundredGigabitEthernet7/0/0       1     down       9000/0/0/0
HundredGigabitEthernet8/0/0       2     down       9000/0/0/0
HundredGigabitEthernet9/0/0/4096  3     down       9000/0/0/0
local0                            0     down        0/0/0/0
```

Next, configure the IP address and interface state by using the `VPP_IF#_NAME` variable that was set before.

```
ubuntu@vpp-node:~$ alias vppctl="docker compose exec vpp vppctl"
ubuntu@vpp-node:~$ vppctl set int ip address $VPP_IF1_NAME 192.168.0.2/24
ubuntu@vpp-node:~$ vppctl set interface state $VPP_IF1_NAME up
ubuntu@vpp-node:~$ vppctl set int ip address $VPP_IF2_NAME 192.168.1.2/24
ubuntu@vpp-node:~$ vppctl set interface state $VPP_IF2_NAME up
```

Finally, return to Pktgen-DPDK and check if the receiving rate has increased by approximately 5.8 times compared to the original test, to 99.2 Gbps (101543 Mbps / 1024).

```
0-1 of 2   <Main Page>  Copyright(c) <2010-2023>, Intel Corporation
  Port:Flags        : 0:P------      Single 1:P--I---      Single
Link State          :       <UP-100000-FD>        <UP-100000-FD>
---Total Rate---
Pkts/s Rx           :                   0              1407830
1407830
       Tx           :             1408000                    0
1408000
MBits/s Rx/Tx       :           0/101556             101543/0
101543/101556
...
```

2.7.3. Hands-on with Netronome Agilio CX

This tutorial provides a basic introduction to P4 programming on Netronome SmartNICs Agilio CX 2x10Gb. Here, you will find all the steps for installing and setting up the development environment, as well as presenting, implementing, and running a simple P4 program on a Netronome SmartNIC. The goal is to provide a quick learning curve by covering only the essential topics up to the implementation of the first program. To get complete step-by-step instructions, please refer to our public repository.

2.7.3.1. Installation and Environment Setup

To install the necessary drivers and modules and configure the environment, follow the instructions below, as root:

```
smartness@host# cd ~/
smartness@host# git clone https://github.com/ \
                guimvmatos/SBRC24NetronomeTutorial.git
```

After this, access the Agilio-P4-SmartNIC directory and follow the tutorial instructions to install the required drivers and modules. By following these steps, you will be ready to start developing and running your P4 programs on Netronome SmartNIC. In the installation directory, you will find a lot of Netronome related documents, that can be useful in case you want to deep into this technology.

2.7.3.2. Details about Netronome architecture

The Netronome SmartNIC uses single-root input/output virtualization (SRIOV), which enables virtual functions (VFs) to be created from a physical function (PF). The VFs thus share the resources of a PF, while VFs remain isolated from each other. The isolated VFs are typically assigned to virtual machines (VMs) on the host. This way, the VFs allow the VMs to directly access the PCI device, bypassing the host kernel. In this tutorial, we have two physical (`p0`, `p1`) and four virtual interfaces (`Vf0.0` to `Vf0.3`). We will work with a P4 program that implements simple IPv6 forwarding, which can be found at `IPv6Forwarding`.

2.7.3.3. Deployment

Here we will show how to deploy, configure, and debug your programs on Netronome SmartNICs. We are taking into consideration that you already clone this repository and that your environment is configured. Once you have your Netronome configured on your machine, you need to locate the `src/p4-16` folder inside the `Agilio-P4-SmartNIC` directory. To get started, ensure you are inside the `p4-16` folder, and then create a folder called `SimpleIPv6`. After that, copy the contents of the `IPv6Forwarding` folder into the folder you just created:

```
smartness@host# cd /root/SBRC24NetronomeTutorial/Agilio-P4-SmartNIC/src/p4-16
smartness@host# mkdir SimpleIPv6 \&\& cd SimpleIPv6/
smartness@host# cp ../../../../IPv6Forwarding/* ./
```

For P4 programming using Netronome, these files are all that is needed. The `ipv6_forwarding.p4` file represents our program itself. The `user_config.json` is the file that will populate the control plane tables. As this tutorial is for those who are already familiar with the P4 programming language, we will not go into details. However, a very common issue for those who are starting to program using Netronome SmartNICs is how to perform interface assignments in the control plane table. When we are working with Netronome SmartNICs, we have physical and virtual interfaces, and you can configure this setting by editing `/lib/systemd/system/nfp-sdk6-rte.service`. Locate and change the following line.

```
Environment=NUM_VFS=4
```

With this configuration, you will have 4 virtual interfaces called VFs. And they can be instantiated in the control plane tables as `v0.0`, `v0.1`, `v0.2`, and `v0.3`. If you want to use the physical interfaces, you should refer to them as `p0` and `p1`. In the `user_config.json` file, we can find examples of how to populate the control plane tables using the aforementioned nomenclature. Now that we have copied the two necessary files, we should call the compiler passing the P4 file as a parameter to create firmware. Then, we will deploy the firmware to the board, and finally, we will populate the control plane tables. To do this, execute the following commands:

```
smartness@host# sudo ./opt/netronome/p4/bin/nfp4build
                     --nfp4c\p4\_version 16
                     --no-debug-info -p out -o firmware.nffw
                     -l lithium -4 ipv6\_forward.p4

smartness@host# sudo ./opt/netronome/p4/bin/rtecli design-load
                    -f firmware.nffw -p out/pif\_design.json

smartness@host# sudo /opt/netronome/p4/bin/rtecli \
                     config-reload -c user\_config.json
```

If you encounter issues with initializing the service or deploying the program, you can find the logs in `/var/log/nfp-sdk6-rte.log` and better understand what is happening. If you want, you can check the configured rules about the control plane configuration.

```
smartness@host#sudo ./opt/netronome/p4/bin/rtecli tables
                    -i 0 list-rules
```

Now that your board is properly configured, to run the test programs, open more two shells, navigate to `/Agilio-P4-SmartNIC/src/p4-16/SimpleIPv6` directory, and in the first shell, run the command `python3 receive.py`. The program will execute and wait for any packet received on interface `v0.3`. After that, in the second terminal, run `python3 send_pkt.py`. A packet will be sent on interface v0.0 destined for v0.3. If the execution of the programs was successful, congratulations! You have completed this tutorial, and the program is ready for the next step. If you are interested in delving deeper and performing more advanced functions, please refer to our public repository. There, you will also find a tutorial for deploying a more advanced program, using SRv6 to connect traffic from multiple virtual machines.

Table 2.2. An overview of recent scientific publications featuring SmartNICs.

Year	Conference	Paper	SmartNIC model	Architecture
2023	SIGCOMM	Unleashing SmartNIC Packet Processing Performance in P4	Nvidia BlueField2 (2 ports×100Gbps).	SoC
2023	SIGCOMM	Lightning: A Reconfigurable Photonic-Electronic SmartNIC for Fast and Energy-Efficient Inference	Nvidia A100X DPU	SoC
2023	SIGCOMM	LEED: A Low-Power, Fast Persistent Key-Value Store on	Mellanox ConnectX-5 NIC	ASIC
2023	SIGCOMM	ClickINC: In-network Computing as a Service in Heterogeneous Programmable Data-center Networks	Netronome smartNIC	ASIC
2023	SIGCOMM	Direct Telemetry Access	Mellanox Bluefield-2 DPU	SoC
2023	SIGCOMM	DBO: Fairness for Cloud-Hosted Financial Exchanges	Nvidia ConnectX-5 NIC	ASIC
2023	SIGCOMM	BMW Tree: Large-scale, High-throughput and Modular PIFO Implementation using Balanced Multi-Way Sorting Tree	Xilinx Alveo U200	FPGA
2023	SIGCOMM	NeoBFT: Accelerating Byzantine Fault Tolerance Using Authenticated In-Network Ordering	Xilinx Alveo U50 FPGA	FPGA
2023	CONEXT	SPADA: A Sparse Approximate Data Structure representation for data plane per-flow monitoring	Xilinx Alveo U280	FPGA
2023	NSDI	SRNIC: A Scalable Architecture for RDMA NICs	Xilinx FPGA	FPGA
2023	NSDI	Rearchitecting the TCP Stack for I/O-Offloaded Content Delivery	Mellanox BlueField-2	SoC
2023	NSDI	Waverunner: An Elegant Approach to Hardware Acceleration of State Machine Replication	Xilinx U280 FPGA	FPGA
2023	NSDI	LASH: Towards a High-performance Hardware Acceleration Architecture for Cross-silo Federated Learning	Xilinx VU13P FPGA	FPGA
2023	NSDI	ExoPlane: An Operating System for On-Rack Switch Resource Augmentation	Netronome Agilio 40 Gbps smart NICs	ASIC
2023	NSDI	RingLeader: Efficiently Offloading Intra-Server Orchestration to NICs	100G Alveo U280 Data Center Accelerator Card	FPGA
2022	SIGCOMM	Predictable vFabric on Informative Data Plane	Xilinx Alveo U200 card,	FPGA
2022	SIGCOMM	Implementing ChaCha Based Crypto Primitives on Programmable SmartNICs	Netronome Agilio CX 40 Gbit/s	ASIC
2022	CONEXT	PipeDevice: A Hardware-Software Co-Design Approach to Intra-Host Container Communication	Intel Arria 10 FPGA	FPGA
2022	NSDI	FlexTOE: Flexible TCP Offload with Fine-Grained Parallelism	Netronome Agilio CX40 40 Gbps	ASIC
2022	NSDI	Re-architecting Traffic Analysis with Neural Network Interface Cards	Netronome Agilio CX, with an NFP4000	ASIC
2022	NSDI	Gearbox: A Hierarchical Packet Scheduler for Approximate Weighted Fair Queuing	Xilinx Alveo U250	FPGA
2022	NSDI	Enabling In-situ Programmability in Network Data Plane: From Architecture to Language	Xilinx Alveo U280	FPGA
2022	NSDI	An edge-queued datagram service for all datacenter traffic	Mellanox BlueField 2)	SoC
2022	NSDI	Buffer-based End-to-end Request Event Monitoring in the Cloud	Broadcom PS225 SmartNICs	SoC
2022	NSDI	Yeti: Stateless and Generalized Multicast Forwardin	NetFPGA SUME [	FPGA
2022	NSDI	solation Mechanisms for High-Speed Packet-Processing Pipeline	Xilinx Alveo U250 board	FPGA
2022	NSDI	Tiara: A Scalable and Efficient Hardware Acceleration Architecture for Stateful Layer-4 Load Balancing	Xilinx FPGA-based SmartNIC	FPGA
2021	SIGCOMM	Gimbal: Enabling Multi-tenant Storage Disaggregation on SmartNIC JBOFs	Mellanox ConnectX-5 NIC	ASIC
2021	SIGCOMM	A Cloud-Scale Per-Flow Backpressure System via FPGA-Based Heavy Hitter Detection	Xilinx's vu9p	FPGA
2021	SIGCOMM	CAMES: Enabling Centralized Automotive Embedded Systems with Time-Sensitive Network	FPGA	FPGA
2021	SIGCOMM	NanoTransport: A Low-Latency, Programmable Transport Layer for NICs	Chisel FPGA	FPGA
2021	SIGCOMM	CocoSketch: High-Performance Sketch-based Measurement over Arbitrary Partial Key Query	Xilinx Alveo u280	FPGA
2021	SIGCOMM	SiP-ML: High-Bandwidth Optical Network Interconnects for Machine Learning Training	Stratix V FPGAs	FPGA
2021	SIGCOMM	Concordia: Teaching the 5G vRAN to Share Compute	FPGA (Terasic DE5-Net) f	FPGA
2021	SIGCOMM	wo beams are better than one: Towards Reliable and High Throughput mmWave Links	Artix-7 FPGA	FPGA
2021	SIGCOMM	1Pipe: Scalable Total Order Communication in Data Center Networks	Mellanox ConnectX-4	ASIC
2021	NSDI	BMC: Accelerating Memcached using Safe In-kernel Caching and Pre-stack Processing	Netronome Agilio CX 2x40GbE	ASIC
2021	NSDI	Flightplan: Dataplane Disaggregation and Placement for P4 Programs	Xilinx ZCU102 FPGA	FPGA
2021	NSDI	ATP: In-network Aggregation for Multi-tenant Learning	Mellanox ConnectX-5	ASIC
2021	NSDI	CodedBulk: Inter-Datacenter Bulk Transfers using Network Coding	Xilinx Virtex-7 XC7VX690T FPGA	FPGA
2021	NSDI	Verification and Redesign of OFDM Backscatter	Microsemi AGLN250 low power FPGA	FPGA
2021	NSDI	Simplifying Backscatter Deployment: Full-Duplex LoRa Backscatter	AGLN250 Igloo Nano FPGA	FPGA
2020	SIGCOMM	An Artifact Evaluation of NDP	NetFPGA	FPGA
2020	SIGCOMM	Using Deep Programmability to Put Network Owners in Control	Xilinx SmartNIC	SoC
2020	SIGCOMM	Defending lightweight virtual switches from cross-app poisoning attacks with vifc	netFPGA	FPGA
2020	SIGCOMM	Optimized Tracing of iCF-enabled Programmable Data Planes	netFPGA	FPGA
2020	CONEXT	DeepMatch: Practical Deep Packet Inspection in the Data Plane using Network Processor	Netronome	ASIC
2020	NSDI	NetTLP: A Development Platform for PCIe devices in Software Interacting with Hardware	Xilinx Kintex 7 FPGA	FPGA
2020	NSDI	TinySDR: Low-Power SDR Platform for Over-the-Air Programmable IoT Testbeds	FPGA (not specified)	FPGA
2020	NSDI	Enabling Programmable Transport Protocols in High-Speed NICs	Kintex Ultrascale+ XCKU15P FPGA	FPGA
2020	NSDI	AccelTCP: Accelerating Network Applications with Stateful TCP Offloading	Netronome Agilio LX 40GbE NIC	ASIC
2020	NSDI	VMscatter: A Versatile MIMO Backscatter	FPGA Microsemi Igloo Nano AGLN250	FPGA

2.8. Challenges and Future Trends

Standartization of SmartNICs. Despite existing efforts towards defining a common NIC architecture [(PNA) 2023] and programming framework such as P4 language, we are still far from having standard programming interfaces fully available in all SmartNICs. As we have seen in previous sections, there are multiple ways to program SmartNICs. The lack of proper programming standards across multiple SmartNIC vendors hinders the wide adoption of network offloaded solutions and suffers from interoperability hazards amongst hardware platforms.

Variable Performance. Packet processing performance in SmartNIC is subject to significant variation across programs, traffic types, and table entries – and, therefore, line-speed processing is not an automatic guarantee. There are two reasons for that. As discussed, SmartNICs usually follow a processing model where a packet is assigned to a particular processing engine in a run-to-completion manner. Second, existing P4 compilers focus on switch ASICs, where resource constraints are the first-order concern, and performance is guaranteed as long as the packed program fits inside the device. [Xing et al. 2023] argue that P4 compilers need to be revised in order to consider SmartNIC architecture nature. A few studies have been done to understand the performance of SmartNICs offloaded programs (e.g., [Viegas et al. 2021, Katsikas et al. 2021]). However, programmers still need to know hardware platform details to extract the most out of the offloaded program. In this context, automatic performance estimation of SmartNIC code would help programmers reduce the time developing/testing applications.

Limited Resource Orchestration. SmartNIC can run different offloaded codes from different applications/tenants. That can be achieved using SoC cores, or specialized-ASIC processing units. Despite a few related studies [Saquetti et al. 2020], little has yet been done to provide a unified resource orchestration layer for SmartNICs, allowing to run multiple isolated applications on top of the same platform. NVIDIA BlueField, for instance, allows running multiple DOCA applications in the SoC. However, there is little support to isolate them with current virtualization techniques. Others vendors, such as Netronome, do not provide any support for isolation, or resource orchestration. Therefore, as SmartNICs follow a run-to-completion model, an application can interfere with the performance of others running on the same card.

Limited Programmability. Each architecture provides a set of programmable primitives and constraints. For instance, Netronome provided limited coherent access to external memory and a lack of a complex arithmetic logic unit. These constraints lead to some workarounds, like employing fixed-point representations of real numbers. These methods can constrain the accuracy of the computing performed in the data plane. In turn, FPGA-based SmartNICs can provide rich hardware primitives as far as the hardware description can be synthesized. In addition to the aforementioned limited programmability, debugging/troubleshooting them also requires additional efforts from programmers.

Application offloading. SmartNICs and hardware accelerations enable a plethora of sys-

tem optimizations. However, it is still not clear which application to offload and, more importantly, which parts of the application to offload. In part, this is due to the existing hardware limitations (e.g., not all applications benefit from running in a SmartNIC), and the diversity of existing hardware architectures and programming tools. Defining a methodology or a guideline is crucial to ensure an optimal matching between the application's needs and networking accelerators. Taking a packet processing application, it is hard to grasp which part (if any) is supposed to be offloaded, and more importantly to which acceleration technology: hardware SmartNIC, or DPDK, or XDP/eBPF, or a combination of them. For example, [Xing et al. 2023] proposes to use Profile-Guided Optimization to tailor P4 code to the underlying SmartNIC hardware enabling better performance optimizations by leveraging knowledge about the input. Despite this effort, we still need better development tools (e.g., specific compilers) to automatically support network technology developers.

Opportunities with Compute eXpress Link (CXL). The Compute Express Link (CXL) is an open industry-standard interconnect between processors and devices such as accelerators, memory buffers, smart network interfaces, persistent memory, and solid-state drives. Since its first release in 2019, CXL has evolved through three generations. CXL offers coherency and memory semantics with bandwidth that scales with PCIe capacity while achieving significantly lower latency than PCIe [Li et al. 2023]. In short, CXL focus on tackling the following challenges: (i) coherent access to system and device memory; (ii) memory scalability; (iii) inefficient usage of CPU/memory due to stranded resources; and (iv) fine-grained data sharing in distributed systems. Non-coherent accesses work well for streaming I/O operations such as storage access. In the context of networking accelerators (e.g., FPGAs or SoCs), entire data structures are moved from system memory to the accelerator for specific functions before being moved back to the main memory and software mechanisms are used to avoid simultaneous accesses between CPUs and accelerators. The usage of CXL will enable a multitude of optimizations concerning packet processing both in hardware and software. That will demand efficient hardware-software co-design to explore the full potential of CXL.

2.9. Closing Remarks

The emergence of programmable network data plane technologies, particularly SmartNICs, has drastically changed the way network operations and management are handled. By deploying custom networking solutions directly within network devices, operators can have more control and make per-packet forwarding decisions at incredibly high speeds.

The growing interest from both academic and industrial sectors in SmartNICs highlights their potential to revolutionize network performance and efficiency. Leading companies such as Nvidia, Netronome, Intel, AMD, and Xilinx are competing to offer hardware solutions that can offload complex networking tasks from host CPUs. This enhances packet processing capabilities while lowering overall ownership costs.

The evolution of NICs into SmartNICs represents a journey towards greater programmability and specialization, driven by the escalating demands of modern networking environments. However, realizing the full potential of SmartNICs requires grappling with

challenges related to programming, debugging, and operating these devices efficiently. Furthermore, understanding the intricacies of SmartNIC architectures and their programming ecosystems is essential for designing and deploying cutting-edge in-network solutions. This chapter provides foundational insights into SmartNIC design principles, hardware architectures, programming languages, and performance considerations. Additionally, it offers a hands-on tutorial featuring state-of-the-art SmartNICs, enabling practitioners to delve deeper into the practical aspects of leveraging this transformative technology.

As SmartNICs continue to evolve and permeate various networking domains, their impact on network performance, scalability, and innovation is poised to be profound. By embracing and mastering SmartNIC technologies, network practitioners can unlock new possibilities for delivering tailored, efficient, innovative, and evolvable networking solutions.

Acknowledgments

This study was partially funded by CAPES, Brazil - Finance Code 001. This work was partially supported by the Innovation Center, Ericsson S.A., and by the Sao Paulo Research Foundation (FAPESP), grant 2021/00199-8, CPE SMARTNESS.

References

[Barach et al. 2018] Barach, D., Linguaglossa, L., Marion, D., Pfister, P., Pontarelli, S., and Rossi, D. (2018). High-speed software data plane via vectorized packet processing. *IEEE Communications Magazine*, 56(12):97–103.

[Bosshart et al. 2014a] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014a). P4: programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

[Bosshart et al. 2014b] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014b). P4: programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

[Cannarozzo et al. 2024] Cannarozzo, L., Morais, T. B., Lorenzon, A. F., de Souza, P. S. S., Gobatto, L. R., Lamb, I. P., Duarte, P. A. P. R., Azambuja, J. R. F., Lorenzon, A. F., Rossi, F. D., Cordeiro, W., and Luizelli, M. C. (2024). Spinner: Enabling in-network flow clustering entirely in a programmable data plane. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2024)*, pages 1–9.

[Chole et al. 2017] Chole, S., Fingerhut, A., Ma, S., Sivaraman, A., Vargaftik, S., Berger, A., Mendelson, G., Alizadeh, M., Chuang, S.-T., Keslassy, I., Orda, A., and Edsall, T. (2017). Drmt: Disaggregated programmable switching. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 1–14, New York, NY, USA.

[DPDK 2024] DPDK (2024). Internet.

[Guo et al. 2023] Guo, Z., Lin, J., Bai, Y., Kim, D., Swift, M., Akella, A., and Liu, M. (2023). Lognic: A high-level performance model for smartnics. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, page 916–929, New York, NY, USA.

[He et al. 2023] He, Y., Wu, W., Le, Y., Liu, M., and Lao, C. (2023). A generic service to provide in-network aggregation for key-value streams. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 33–47, New York, NY, USA.

[Intel 2021] Intel (2021). P416 intel tofino native architecture—public version. `https://github.com/barefootnetworks/Open-Tofino`. accessed July 20, 2023.

[Katsikas et al. 2021] Katsikas, G. P., Barbette, T., Chiesa, M., Kostić, D., and Maguire Jr, G. Q. (2021). What you need to know about (smart) network interface cards. In *International Conference on Passive and Active Network Measurement*, pages 319–336. Springer.

[Kianpisheh and Taleb 2023] Kianpisheh, S. and Taleb, T. (2023). A survey on in-network computing: Programmable data plane and technology specific applications. *IEEE Communications Surveys Tutorials*, 25(1):701–761.

[Li et al. 2023] Li, H., Berger, D. S., Hsu, L., Ernst, D., Zardoshti, P., Novakovic, S., Shah, M., Rajadnya, S., Lee, S., Agarwal, I., Hill, M. D., Fontoura, M., and Bianchini, R. (2023). Pond: Cxl-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 574–587, New York, NY, USA.

[Liu et al. 2019] Liu, M., Cui, T., Schuh, H., Krishnamurthy, A., Peter, S., and Gupta, K. (2019). Offloading distributed applications onto smartnics using ipipe. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 318–333, New York, NY, USA.

[Min et al. 2021] Min, J., Liu, M., Chugh, T., Zhao, C., Wei, A., Doh, I. H., and Krishnamurthy, A. (2021). Gimbal: Enabling multi-tenant storage disaggregation on smartnic jbofs. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 106–122, New York, NY, USA.

[Moon et al. 2020] Moon, Y., Lee, S., Jamshed, M. A., and Park, K. (2020). {AccelTCP}: Accelerating network applications with stateful {TCP} offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 77–92.

[Njavro et al. 2022] Njavro, A., Tau, J., Groves, T., Wright, N. J., and West, R. (2022). A dpu solution for container overlay networks. *arXiv preprint arXiv:2211.10495*.

[NVIDIA 2024] NVIDIA (2024). Internet.

[Olteanu et al. 2022] Olteanu, V., Eran, H., Dumitrescu, D., Popa, A., Baciu, C., Silberstein, M., Nikolaidis, G., Handley, M., and Raiciu, C. (2022). An edge-queued datagram service for all datacenter traffic. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 761–777.

[OpenNIC 2024] OpenNIC (2024). OpenNIC.

[Pfaff et al. 2015] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., and Casado, M. (2015). The design and implementation of open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA.

[(PNA) 2023] (PNA), P. P. N. A. (2023). Internet.

[Ponomarev and Ghose 1998] Ponomarev, D. and Ghose, K. (1998). A comparative study of some network subsystem organizations. In *Proceedings. Fifth International Conference on High Performance Computing (Cat. No. 98EX238)*, pages 436–443.

[Saquetti et al. 2020] Saquetti, M., Bueno, G., Cordeiro, W., and Azambuja, J. R. (2020). P4vbox: Enabling p4-based switch virtualization. *IEEE Communications Letters*, 24(1):146–149.

[Saquetti et al. 2021] Saquetti, M., Canofre, R., Lorenzon, A. F., Rossi, F. D., Azambuja, J. R., Cordeiro, W., and Luizelli, M. C. (2021). Toward in-network intelligence: Running distributed artificial neural networks in the data plane. *IEEE Communications Letters*, 25(11):3551–3555.

[Schuh et al. 2021] Schuh, H. N., Liang, W., Liu, M., Nelson, J., and Krishnamurthy, A. (2021). Xenic: Smartnic-accelerated distributed transactions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 740–755, New York, NY, USA.

[Swamy et al. 2022] Swamy, T., Rucker, A., Shahbaz, M., Gaur, I., and Olukotun, K. (2022). Taurus: A data plane architecture for per-packet ml. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 1099–1114, New York, NY, USA.

[Viegas et al. 2021] Viegas, P. B., de Castro, A. G., Lorenzon, A. F., Rossi, F. D., and Luizelli, M. C. (2021). The actual cost of programmable smartnics: Diving into the existing limits. In *International Conference on Advanced Information Networking and Applications*, pages 181–194. Springer.

[Wang et al. 2023] Wang, T., Lin, J., Antichi, G., Panda, A., and Sivaraman, A. (2023). Application-defined receive side dispatching on the nic. *arXiv preprint arXiv:2312.04857*.

[Wei et al. 2023a] Wei, X., Cheng, R., Yang, Y., Chen, R., and Chen, H. (2023a). Characterizing off-path SmartNIC for accelerating distributed systems. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 987–1004, Boston, MA.

[Wei et al. 2023b] Wei, X., Cheng, R., Yang, Y., Chen, R., and Chen, H. (2023b). Characterizing off-path SmartNIC for accelerating distributed systems. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 987–1004, Boston, MA.

[Wei et al. 2023c] Wei, X., Cheng, R., Yang, Y., Chen, R., and Chen, H. (2023c). Characterizing off-path {SmartNIC} for accelerating distributed systems. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 987–1004.

[Xavier et al. 2021] Xavier, B. M., Guimarães, R. S., Comarela, G., and Martinello, M. (2021). Programmable switches for in-networking classification. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE.

[Xilinx 2024] Xilinx (2024). Internet.

[Xing et al. 2023] Xing, J., Qiu, Y., Hsu, K.-F., Sui, S., Manaa, K., Shabtai, O., Piasetzky, Y., Kadosh, M., Krishnamurthy, A., Ng, T. S. E., and Chen, A. (2023). Unleashing smartnic packet processing performance in p4. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 1028–1042, New York, NY, USA.

[Yao et al. 2023] Yao, R., Zhang, Z., Fang, G., Gao, P., Liu, S., Fan, Y., Xu, Y., and Chao, H. J. (2023). Bmw tree: Large-scale, high-throughput and modular pifo implementation using balanced multi-way sorting tree. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 208–219, New York, NY, USA.

[Zhang et al. 2023] Zhang, J., Cheng, X., Wang, W., Yang, L., Hu, J., and Chen, K. (2023). {FLASH}: Towards a high-performance hardware acceleration architecture for cross-silo federated learning. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1057–1079.

This page is intentionally left blank.

Chapter 3

Generation of Synthetic Datasets in the Context of Computer Networks using Generative Adversarial Networks

Thiago Caproni Tavares (IFSULDEMINAS), Ariel Góes de Castro (UNICAMP), Leandro C. de Almeida (IFPB), Washington Rodrigo Dias da Silva (UFSCAR), Christian Esteve Rothenberg (UNICAMP) and Fábio Luciano Verdi (UFSCAR)

Abstract

This chapter explores the domain of generative artificial intelligence, focusing on its transformative impact in producing network datasets. Our tutorial delves into Generative Adversarial Networks (GANs), initially pivotal in image and media synthesis but now extended to computer networking, offering nuanced capabilities in generating synthetic data and facilitating data-driven predictions, classifications, and experimentation. This work discusses the utility of GANs in computer networks, specifically in generating synthetic data while preserving privacy and enhancing dataset representativeness. The tutorial section of this book chapter elucidates on generating synthetic time series data using GANs, with practical applications ranging from telemetry data generation to creating synthetic Packet Capture (PCAP) files. Further contributions of this work include exploring the use of synthetic data in training Reinforcement Learning (RL) agents for resource optimization, highlighting the broader implications and emerging trends in applying generative AI in computer networks.

3.1. Introduction and Motivation

Generative artificial intelligence encompasses a suite of algorithms and models endowed with the capacity to produce diverse forms of data, including images, videos, text, and assorted digital media. In contemporary times, this paradigm has gained noteworthy traction among individuals outside academic circles, attributed mainly to the advent of ChatGPT [34], a prominent instance of a Large Language Model (LLM). This approach establishes a mechanism for comprehending and engendering human language expressions. Moreover, it represents an evolutionary progression from conventional Language Mod-

els, characterized by integrating substantial parameter magnitudes that yield exponential enhancements in learning capabilities.

An additional category of generative models is represented by Generative Adversarial Networks (GANs). Primarily introduced within the context of image synthesis, GAN has garnered considerable attention due to its efficacy in capturing intricate and high-dimensional data distributions. This ability is realized through two distinct Neural Networks — an entity designated as the generator and another as the discriminator — that engage in an interplay following the principles of game theory, as delineated in [15]. The fundamental operational paradigm involves the generator network producing synthetic data samples to deceive the discriminator. In parallel, the discriminator network undertakes the role of a judge, assessing the similarity between real and generated/synthetic data. The main objective is to generate a scenario wherein the discriminator's capacity to discern actual data from its synthetic counterparts is markedly diminished.

In this context, GAN has seen an extension of its application into the domain of Computer Networks over recent years. Navidan et al. [38] have undertaken a classification of GAN based on their specific application objectives. To illustrate, GANs can generate synthetic data to serve as inputs for distinct learning models. Alternatively, GANs may allocate one of their constituent neural networks—the generator or the discriminator—toward data prediction or classification tasks, respectively.

The utilization of GANs in the context of Computer Networks is contingent upon the specifics of the particular application. When employed as a synthetic data generator, GANs work as a simulator. Within this context, the synthetic data produced emulates the inherent distribution of the original dataset, thereby ensuring the preservation of privacy considerations. Moreover, these networks prove instrumental in tasks such as dataset augmentation and balancing, culminating in a dataset characterized by enhanced representational capacity. Consequently, the resultant model emerges as a conduit to share intricate dynamics of real environments while obfuscating inherent complexities and maintaining data quality integrity.

Within computer networks, the application of machine learning manifests across distinct modalities: direct deployment within real setups, engagement within simulated environments, or application of GANs to emulate specific environmental attributes. One of the advantages of GAN is its ability to imitate and embody the inherent characteristics of a given system, thereby creating a closer approximation to reality. This proficiency augments the efforts of both industrial practitioners and researchers, enabling the construction of experimental frameworks guided toward the intelligent orchestration of resources. [16] presents a group created in October 2023 by ATIS Alliance that aims to survey generative AI/ML use cases across the network, demonstrating the importance of generative AI in the context of Computer Networks.

This tutorial book chapter attempts to elucidate the fundamental principles of generating synthetic time series data using GANs. Furthermore, it will delve into the practical applications of GANs in generating telemetry data derived from a Programmable Data Plane (PDP) and creating Packet Captures (PCAPs). This tutorial showcases the diverse utility of synthetic data in various contexts, such as training an RL agent with the objective of PDP resource optimization and generating realistic PCAPs.

This chapter is structured as follows. Section 3.2 delves into the core concepts of GANs, and Section 3.3 explores their applications in In-band Network Telemetry (INT) and PDPs. Section 3.4 covers synthetic data generation's theoretical and practical aspects, including creating telemetry data and synthetic PCAP files. Section 3.5 discusses two use cases where synthetic data can be applied. Finally, Section 3.6 summarizes key findings and discusses emerging synthetic data generation trends and applications.

3.2. Fundamentals of Generative Adversarial Networks

This section provides a foundational overview of Generative Artificial Intelligence, introducing the core concepts and presenting a small literature survey. Section 3.2.1 delves into the intricacies of GANs, elucidating their operational principles and demonstrating their application in training RL agents. Section 3.2.2 covers the fundamental principles of PCAP generation, outlining the methodologies and techniques employed in creating synthetic network traffic data for analysis and testing purposes.

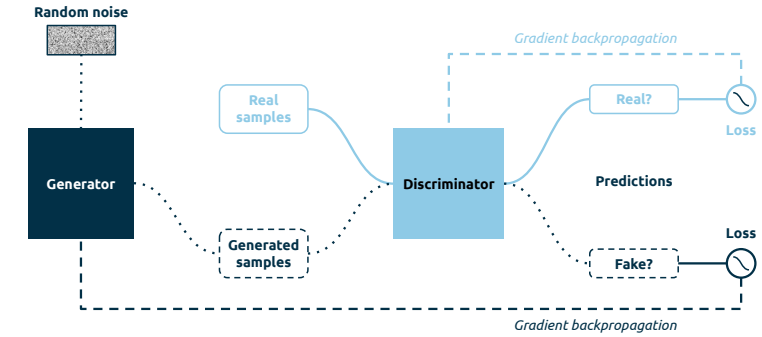
3.2.1. GANs and RL

Machine learning is commonly categorized into three paradigms: supervised, unsupervised, and RL [48]. As an illustration, the well-established GAN predominantly pertains to unsupervised learning. Within this framework are two competing modules: the generator and the discriminator. The first tries to produce synthetic data that mimics the actual data distribution, while the second tries to distinguish between real and synthetic data. The generator and the discriminator are trained in an adversarial manner until they reach an equilibrium where the discriminator cannot tell the difference between real and synthetic data (Figure 3.1).

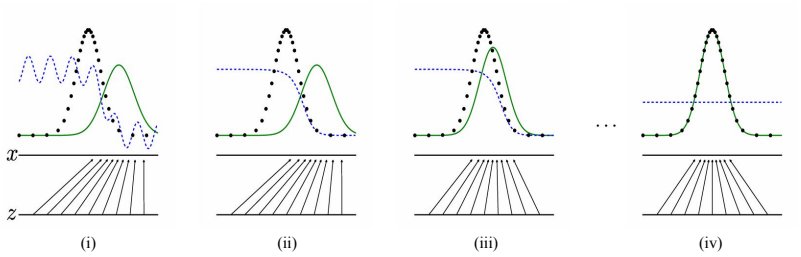
Nevertheless, an alternative perspective emerges by considering the discriminator's role as akin to that of a supervised learning module. This is evident in the discriminator's utilization of actual data instances to ascertain the degree of resemblance between the values generated by the generator and those originating from actual data sources.

GANs have found diverse applications within the ambit of Computer Networks. Their utility spans data generation, system optimization, and data classification, as evidenced by existing literature [56]. The choice of the GAN variant depends on the specific utilization context, which necessitates tailored selections to align with distinct objectives. Although the predominant application of GAN has traditionally been within computer vision, notable strides have been taken towards their adaptation for time series network data. Among the different GAN frameworks enlisted, including Vanilla GAN, Bidirectional GAN (BIGAN), Conditional GAN (CGAN), InfoGAN, CycleGAN, Energy-based GAN (EBGAN), and Least Square GAN (LSGAN), there is an inclination towards encompassing time series data. Recent years have witnessed the ascendancy of specific GAN iterations that have found prominence within the time series domain, including Conditional Tabular GAN (CTGAN), DoppelGANger, and TimeGAN, which are specially tailored to learn the intricacies of time series data [37].

The application of GAN for data generation yields versatile utility across diverse scenarios. In this case, it can effectively address data imbalance concerns, serving to rectify skewed datasets or serve as a mechanism to impute absent values. Its strength



(a)



(b)

Figure 3.1: Adversarial training overview: The training pipeline is shown in figure (a), and an illustration of the process is shown in figure (b). The GAN trains by updating the discriminator function D (blue dashed line) and the generator function G simultaneously, enabling D to differentiate real data samples from the distribution p_x (black dotted line) from generated samples of p_g (green solid line). The z axis represents the input noise, and the x axis represents the real data domain. The arrows illustrate how mapping from z to x imposes non-uniformity (p_g) on transformed samples. Key steps include: (i) showing an adversarial pair near convergence, (ii) illustrating the training of D to discriminate data samples, (iii) after updates, D guides G to produce samples more akin to real data, and (iv) with enough capacity and training steps, G and D might converge to a point where neither can improve, as D cannot differentiate between p_x and p_g. Adapted from [15].

extends to privacy preservation, enabling the obfuscation of individuals' personally identifiable information. Presently, entities have emerged in the corporate landscape offering services tailored to the treatment of sensitive data, facilitating the creation of synthetic data generators that facilitate secure information exchange between organizations. In certain instances, these data generators are pivotal in constructing novel datasets. Such datasets subsequently find application within varying contexts, often in conjunction with complementary machine learning methodologies, such as RL, as demonstrated by [24].

RL is fundamentally characterized as a machine learning paradigm wherein an autonomous agent aims to optimize decision-making processes within an environment that is not known to the agent. Central to the construct of an RL algorithm is the interplay between the agent and the environment (Figure 3.2). A set of states defines the environment, while the agent is endowed with the capacity to execute actions upon this environment.

The outcomes of these agent-initiated actions, in turn, produce either rewards or penalties computed based on the observations derived from the environment. A positive environmental alteration following an action elicits a reward, while a harmful impact triggers a penalty. Thus, the overarching objective within RL is enhancing state-action pair optimization through iterative trial-and-error exploration of the agent's actions [47].

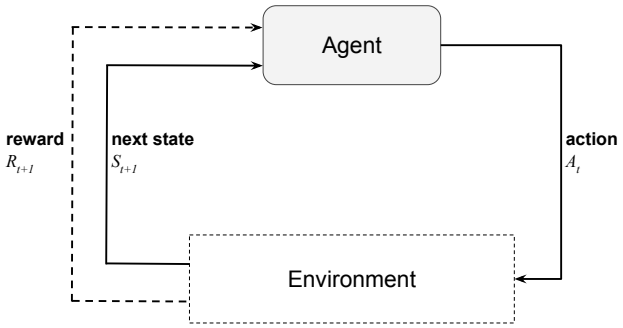


Figure 3.2: Agent-environment interaction. The diagram presented in this figure illustrates a sequential decision-making strategy called Markov Decision Process (MDP). Within this framework, an agent continually interacts with the environment by taking actions (A) at specific time steps (t) and observing subsequent states (S_{t+1}) resulting from those actions. A reward value (R_{t+1}) is generated for each interaction to assess the action effectiveness, which is expected to be maximized throughout the agent's training process. Adapted from [47].

Xiong et al. [49] extensively elucidates a plethora of applications of RL within the purview of mobile networks. These applications encompass domains such as Network Slicing, Power Control in Cellular Networks, Computation Offloading, Edge Caching, and other segments of the mobile network architecture characterized by resource optimization challenges. The adeptness of RL finds resonance, particularly in scenarios where the allocation and utilization of resources are paramount, ultimately culminating in enhanced efficiency and performance across diverse facets of mobile networks [18].

GAN and RL used jointly remain relatively underexplored within the existing literature. A comprehensive exploration highlighting the advantages gleaned from the confluence of these two methodologies is lacking. While RL has garnered substantial traction across many domains, the prospect of its integration with GAN has yet to be examined. Given RL's robustness in optimizing decisions and GAN's capacity to simulate intricate data distributions, this intersection carries considerable potential. GANs can model and generate realistic network traffic patterns, serving as training data for RL agents responsible for testing and optimizing network configurations and policies.

However, an important issue must be considered when using RL in an actual setup compared to a simulated scenario. Sim-to-real discrepancy is an important issue that must be addressed in the context of mobile network research. The concept is defined as the gap between simulation and real environments. It is not a particular problem in the field of communication, as it can also be observed in other domains, such as computer vision, natural language processing, robotic control, and autonomous driving [35]. The layered construction of computer networks increases the difficulty of accurately simulating a real

configuration, complicating the discrepancy between these approaches.

Qiang et al. [29] discussed an automatic online service configuration in network slicing. The authors have proposed and implemented Atlas's solution on an end-to-end network prototype. The system aims to automate the network-slicing process to reduce the Sim-to-Real discrepancy. To do that, they created a methodology divided into three stages, from simulation to real network. The authors proposed using a new parameter-searching method based on Bayesian optimization. Complementary to that work, our study aims to show that it is possible to reduce the Sim-to-Real discrepancy using TimeGAN instead of Bayesian optimization. So, we show the feasibility of this approach by training a RL model and comparing its training using real and synthetic data.

The work presented by Hua et al. [24] introduces a strategy that merges deep RL with distributional modeling to improve the allocation of resources like bandwidth among various network slices. This technique elevates efficiency and service quality within network slicing setups. By harnessing advanced methodologies from RL and distributional modeling, this approach resolves the intricate issues of resource management within complex network-slicing contexts. It's worth noting that while the authors' approach employs a GAN to approximate the distribution of state-action values in an RL model, our method primarily focuses on generating simulated environmental data to facilitate RL training.

Falahatraftar et al. [13] also proposes an approach for addressing network slicing in heterogeneous vehicular networks using CGAN. The main idea involves partitioning a network into multiple virtual networks to cater to different service requirements. The authors suggest employing GANs to generate tailored network slices that match specific vehicular communication needs. This approach aims to optimize resource allocation, enhance communication quality, and enable efficient coexistence of various vehicular applications within the network. However, the authors trained the CGAN model using data from a simulated scenario because of the lack of data sets from real environments.

The integration of GAN and RL techniques to estimate channel coefficients is investigated in [30]. Similar to prior research, the aim is to create synthetic data through GAN and employ it to train RL algorithms, contributing to the refinement of the estimation process. By leveraging GAN's data generation capabilities and RL's adaptable learning, the proposed method enhances the precision of channel coefficient estimation. This underscores the potential of machine learning paradigms for wireless communication applications. However, it's important to note that the authors do not utilize a real experimental setup or environment. Instead, they generate the dataset using Gaussian distributions for GAN training, omitting the use of real data. Obtaining actual network data for these tasks poses significant challenges. First, network data may contain sensitive or private information of the users or organizations, which raises ethical and legal issues for sharing or publishing them. Second, network data may be scarce or outdated [41], especially for emerging or evolving network scenarios (e.g., 5G and beyond). Third, network data may be biased/incomplete [10], which limits the generalization and robustness of the network analysis models.

3.2.2. PCAP Generation

Neural networks offer a powerful solution to the challenges of obtaining and utilizing actual network data for analysis tasks. One of the primary advantages of employing neural networks in this context is their ability to learn complex patterns and relationships from existing network traces, such as PCAPs (Packet Capture files), and augment existing datasets with synthetic packet traces. In this manner, network traces are augmented without user privacy concerns – in addition to the possibility of teaching the model how and/or where to generate anomalies.

Several papers have proposed GAN-based methods for generating different aspects of network traffic, such as packet headers, packet payloads, packet lengths, packet inter-arrival times, flow features, and control-plane messages. To our knowledge, Ring et al.[42] was the first to utilize GAN models to generate network traffic data. They leveraged two GAN variants, WGAN [3] and WGAN-GP [17]. PcapGAN [12] proposes a method for generating realistic PCAP files that preserve the style and structure of real PCAP files. The method uses a style-based GAN architecture that can control the style of the generated packets at different levels of abstraction. The method can also generate packets with protocols, such as TCP, UDP, ICMP, ARP, and DNS. Shahid et al. [44] combined an autoencoder with WGAN and WGAN-GP to generate IoT traffic so it could detect rare attacks. The generated data had data distribution characteristics similar to real data. However, the model had little improvement in detecting rare attacks due to dataset imbalance. SIP-GAN [32] introduces a method for generating realistic SIP (Session Initiation Protocol) traffic that can be used for testing VoIP (Voice over IP) systems. The method uses a conditional GAN architecture that can generate SIP messages with different types, such as INVITE, ACK, BYE, CANCEL, and OPTIONS. The method can also generate SIP messages with different attributes, such as call duration, caller ID, callee ID, and session description. Similarly, proposes a new and generalized two-pass approach to evaluating the quality of samples produced by the generator to produce a filtered, higher-quality output data set.

NetShare [51] and Han et al. [20] leverage Wassertein GAN variations. The first generates synthetic IP header traces that can be used for network analysis and anomaly detection. The method uses a Wasserstein GAN with gradient penalty (WGAN-GP) architecture to generate IP headers with realistic features, such as source IP address, destination IP address, protocol type, port number, and packet length. Also, it preserves the statistical properties of the original traces, such as mean, variance, auto-correlation, and cross-correlation. Han et al. [20] also leveraged synthetic packet payloads that can be used for smart cybersecurity applications. The method uses a flow-based WGAN architecture to generate packet payloads with different lengths and contents. The method also uses an attention model based on byte embedding that can learn the correlations between different bytes in the packet payloads. Similarly, Lee et al. [27] improved the identification accuracy of sparse assaults in network intrusion detection and developed a data augmentation strategy based on the WGAN-GP model.

Soper et al. [45] introduced a two-pass approach that consists of two steps: (i) generating synthetic network traffic data sets using existing methods; (ii) refining the synthetic network traffic data sets using an error correction model based on recurrent neural

networks (RNNs). The method can reduce errors between real and synthetic network traffic data sets regarding statistical properties and classification performance. On the other hand, Buhler et al. [8] generates live network traffic from code repositories. The method uses a code analysis tool to extract network-related information from code files, such as protocol type, port number, packet content, and packet frequency. The method can also use a code execution engine to run the code files and generate live network traffic based on the extracted information. Meng et al. [33] uses a Markov chain model to capture the temporal and spatial patterns of control-plane traffic, such as the state transitions, the inter-arrival times, and the location updates. The method can also use a GAN model to generate synthetic control-plane traffic based on the Markov chain model. Holland et al. [22] introduce nPrint, a tool that generates a unified packet representation amenable to representation learning and model training. They also integrate nPrint with automated machine learning (AutoML), resulting in nPrintML. This public system eliminates feature extraction and model tuning for traffic analysis tasks. The paper evaluates nPrintML on eight separate traffic analysis tasks and shows that it can achieve comparable or better performance than state-of-the-art methods with minimal human intervention. More recently, NetDiffusion [25] took a step forward and treated packets as images. It leverages the nPrint [22] the fixed-length standardized packet representation to transform packets into images that can be easily manipulated, considering the advent of generative models with a customized diffusion model.

This section delved into the fundamental concepts of GANs, providing essential insights into RL, and discussing the generation of PCAP files. Building upon this foundation, Section 3.3 complements the discussion by integrating these concepts with network programmability, examining how the principles outlined in this section apply to and enhance the domain of programmable networks.

3.3. In-band Network Telemetry and Programmable Data Planes

With the emergence of Software Defined Networking [31], the initial advances towards network programmability were taken [21]. The separation of data and control planes has brought about increased flexibility and new avenues of research in computer networks.

In this context, the OpenFlow protocol pioneered the provision of a network abstraction layer to the control element (Controller), thereby enabling the configuration and manipulation of the network's data plane through software programming. However, this model lacked sufficient flexibility for adding new headers and defining new actions after flow matching due to limitations imposed by the rigidity in the intelligence embedded in the pipeline of ASIC processors [14].

Traditionally, the data plane has taken on the role of processing packets according to the logic prescribed by the control plane. However, the game changes with the introduction of data plane programmability, which facilitates the incorporation of intelligence during packet processing at the hardware's most proximate level without the necessity for control plane intervention. Noteworthy examples of languages tailored for programming the data plane in this context include P4 and NPL. Among these, P4 is a language that has gained wide acceptance within scientific and industrial communities.

Beyond the evident flexibility, the response time of the data plane resides in the

order of nanoseconds, while the control plane operates on a scale of seconds or milliseconds [21]. P4 code exhibits versatility across various architectures, each characterized by unique match-action pipeline stages and packet processing attributes. These architectural designs are fundamentally underpinned by an abstraction model, such as PISA (Protocol Independent Switch Architecture), that facilitates mapping instructions and hardware-specific features tailored to individual targets. The P4 language abstracts packet parsing and processing by providing a generalized forwarding model.

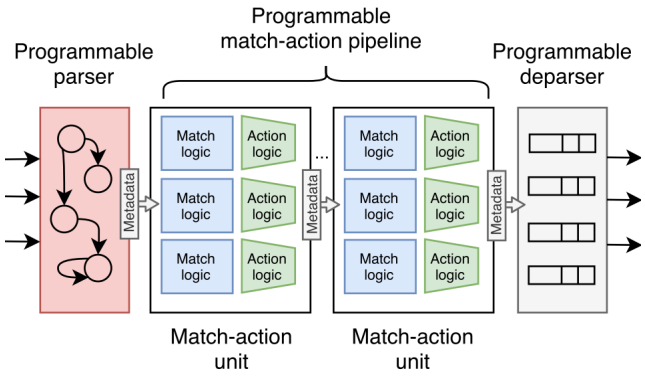


Figure 3.3: PISA abstraction model. Adapted from [21].

As illustrated in Figure 3.3, the PISA architecture consists of three fundamental components: a programmable parser, a programmable match-action pipeline, and a programmable deparser.

The programmable parser operates as a finite state machine that dictates the order of header extraction from incoming network packets. Once the programmable parser has exposed the headers and their associated fields, they can traverse multiple stages within the match-action pipeline. In this context, tables are defined to effectively manage metadata, adhering to the logic established by the programmer. For instance, in IPv4 routing, a table can be configured to perform matches against the destination address and execute actions such as i) decrementing the TTL field, ii) adjusting physical addresses, and iii) configuring the output port. The deparser represents the programmable component that specifies the packet's serialization, i.e., reassembly, for subsequent transmission.

In the context of programmable networking, the industry has recently adopted support for the P4 language. Xilinx has introduced the NetFPGA SUME, an advanced network card enabling P4 programming. Furthermore, industry leaders such as Intel and Broadcom have launched dedicated processors, namely the Tofino and Trident4, which are tailored explicitly for the programmable networking market.

A software-based switch version capable of executing P4 code is available for educational purposes. The Bmv2 (Behavioral Model Version 2) [1] is an integral component within the P4 ecosystem, designed as a tool for the study, testing, and analysis of solutions directly within the data plane.

Each equipment category in the P4 ecosystem, commonly called a "target", possesses a specific architecture. In the case of NetFPGA, the employed architecture is the

[1] `https://github.com/p4lang/behavioral-model`.

Simple Sume Architecture, whereas Tofino utilizes the Tofino Native Architecture (TNA), and the Bmv2 is rooted in the V1model architecture [21].

A comprehensive understanding of the architecture supported by the target is essential for the programmer, as it delineates the implementation particulars of the various stages. Furthermore, the availability of metadata for utilization depends on the specific architecture. These metadata can be accessed and integrated into network monitoring systems. For instance, in the V1model architecture, metadata about switch interface buffers can be retrieved and encapsulated within packets during forwarding. This technique is elaborated in the INT specification [39], designed to facilitate metadata collection within the data plane without necessitating control plane intervention or involvement.

These advancements in programmable data plane have enabled network devices to report the network's state autonomously, eliminating the need for direct control plane intervention [4]. In this scenario, packets incorporate telemetry instructions within their header fields, facilitating the fine-grained collection and recording of network data. The telemetry instructions are defined in the INT data plane specification [39].

Figure 3.4 illustrates the operation of INT within an arbitrary network. The network comprises four hosts, namely *H1, H2, H3*, and *H4*, along with four nodes equipped with P4 and INT support, denoted as *S1, S2, S3*, and *S4*. Each network node possesses a set of metadata, represented by orange (S1), magenta (S2), green (S3), and blue (S4) rectangles. This metadata contains information specific to each node, such as Node ID, Ingress Port, Egress Spec, Egress Port, Ingress Global Timestamp, Egress Global Timestamp, Enqueue Timestamp, Enqueue Queue Depth, Dequeue Timedelta, and Dequeue Queue Depth, as specified in the V1Model architecture.

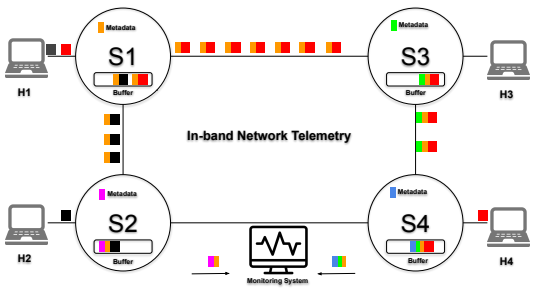


Figure 3.4: INT operation. INT metadata is appended on the packets in each hop. In the specific collection point, the monitoring system receives INT metadata.

Still, in Figure 3.4, two distinct flows are depicted: one represented by red packets and the other by black packets. The red flow is required to adhere to the prescribed network path *f1=H1, S1, S3, S4, H4*, while the black flow must traverse the designated path *f2=H1, S1, S2, H2*.

At each network hop along these paths, the data plane of the network devices employs telemetry instructions to facilitate the collection and inclusion of metadata within the packets as they traverse each node. This process is iteratively performed throughout the path, starting from the first node after the source and concluding at the last node before reaching the destination. Upon reaching the destination node, the metadata is extracted from the packet and relayed to the monitoring system. The original packet is then directed to its final destination.

One of the primary advantages of employing telemetry lies in the exceptional level of granularity it offers. Every packet traversing the network carries pertinent information directly to the monitoring system at the line rate. This level of granularity aligns with the perspective presented in [19], wherein it is recognized that a substantial volume of data can prove helpful for ML algorithms.

However, the dynamic nature of network traffic necessitates ongoing adjustments to machine learning models. RL is employed to tackle this issue, enabling agents to learn from variations in traffic characteristics through trial and error. A pivotal element in this context is the convergence time, which indicates when the intelligent agent has effectively learned a policy and started to implement it consistently. Regardless, abrupt changes in traffic dynamics can extend the time required for convergence or, in severe cases, cause the model to diverge. Using synthetically generated samples through GANs is advantageous for accelerating training and improving convergence, as RL agents can rapidly achieve the desired policy. Thus, Section 3.4 shows the challenges associated with synthetic data generation, contextualizing the operational environment and bringing the pertinent problems around it.

3.4. Synthetic Data Generation

This section outlines the dual-use cases of synthetic network data generation employed within the scope of this tutorial. Section 3.4.1 explains the characterization of an environment, providing a comprehensive description of the problem and the methodology adopted to address it. Furthermore, this section also expounds on the characterization of the dataset and the mechanism proposed for selecting the best model. On the other hand, Section 3.4.2 presents all the steps to generate PCAP traces with NetDiffusion and NetShare, the two main tools currently available for PCAP generation. First, the definition of the packet generation problem is presented. Finally, the choices of each work are presented, justifying the main coding/interpretation choices of network packets and how they deal with issues such as temporal (e.g., packet sequence) and spatial (e.g., packet fields) dependencies.

3.4.1. Generation of Telemetry Data

3.4.1.1. Problem definition

The environment where the dataset was collected in the present tutorial is depicted in Figure 3.5. In this setup, each infrastructure component is represented by an isolated virtual machine interlinked through a P4 programmable data plane network. The CDN was deployed to facilitate an MPEG-DASH service featuring live streaming of a soccer game and a playlist housing the ten most frequently accessed YouTube videos. Load management was executed using WAVE [2] [2], a versatile load generator that orchestrates instances of an application over time. A network architecture is instantiated and consists of three programmable switches where the INT telemetry is collected. Complementing this infrastructure, a Video Client is integrated to provide video metrics. At the top of this configuration, an RL agent operates within the network ecosystem, actively engag-

[2]https://github.com/ifpb/wave.

ing to enhance the user experience by orchestrating alterations to queue sizes within the switches. This RL agent's central goal resides in optimizing resource utilization across the switches, thereby elevating the overall efficiency of the network infrastructure.

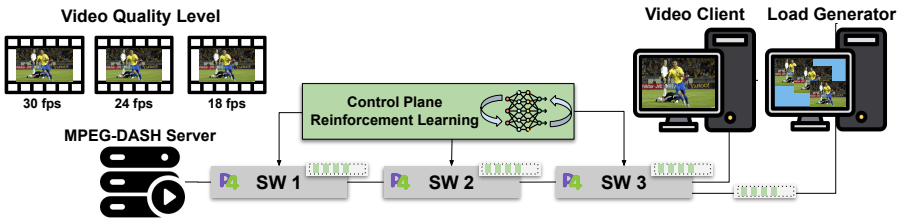


Figure 3.5: Real setup used for the evaluation composed by P4 BMv2 switches, a DASH server and video clients.

The RL model works as a data plane optimizer within this study, orchestrating the management of queue sizes across three distinct switches to enhance user experience. It is important to observe that cooperation from network operators to facilitate experiments of this nature is often challenging. In practice, the necessity arises to devise a mechanism for training the optimizer, or RL agent, without a real setup. In this case, the GAN trained on real data, serving as a simulator replacing the real setup, becomes pertinent. The present work is based on a real setup to train the RL agent while concurrently storing a dataset for employment in a GAN model. Moreover, after an on-the-fly training of the RL model, an offline training of the RL model is executed using synthetic data generated from the TimeGAN. This process facilitates a comparative assessment of the RL agent's generalization capacity across both scenarios: on-the-fly (real setup) and its simulated counterpart realized through utilizing the GAN generator.

However, a pertinent question arises: Given the availability of datasets sourced from operators, is the utilization of GAN to train Machine Learning models, including RL methodologies, necessary? Why not exclusively rely upon the original dataset to train such models?

Collecting a dataset constitutes a prerequisite for training a GAN model, necessitating data acquisition from operators or real setups. However, the inherent dataset may exhibit characteristics such as imbalance, inadequacy, missing or erroneous values, and static nature, preventing the feasibility of repeated experiments and statistical validation as demonstrated in [40]. In contrast, a GAN model, once trained, offers the capacity to generate balanced data devoid of inconsistencies or gaps. Moreover, it facilitates the generation of data volumes requisite for a diverse spectrum of experiments. Additionally, GAN empowers the training of models on different datasets, facilitating the acquisition of varied data distributions to cater to numerous scenarios. Besides, it is also important to highlight the time taken to train an RL model using a synthetic dataset compared to a real scenario. In the context of RL applications, GAN utilization assumes significance in enabling the introduction of diversified scenarios for the agent to learn from, thereby enhancing the agent's capacity to adapt to broader conditions.

Consequently, this tutorial delineates a methodology wherein the efficacy of an RL model is evaluated through its deployment in conjunction with data generated by a

TimeGAN model. The TimeGAN is trained to use telemetry data from video applications within the programmable data planes framework. The following exploration thereby offers insights into the viability and performance of RL when trained with synthetic data generated by TimeGAN.

3.4.1.2. Methodology

Broadly, the methodology adopted in this tutorial is encapsulated within Figure 3.6, and its implementation can be defined through a delineation into four distinct steps:

1. ***Creation of a Real Setup and Data Collection:*** In this preliminary phase, the establishment of a tangible real setup is undertaken, as presented in Sec. 3.4.1.1.
2. ***Dataset Recording and TimeGAN Model Training:*** An ensemble of pertinent features is stored within this stage. Among these features are switch telemetry metrics, notably encompassing switch queue sizes and packet arrival times. Simultaneously, video metrics, such as frames per second (FPS) and resolution, are captured. The ascertained dataset comprises two distinct scenarios, characterized by dissimilar queue sizes—32 and 64 packets—affording coverage of two operational settings.
3. ***Reinforcement Learning Training Utilizing Synthetic Data:*** After training the TimeGAN model, the subsequent step encompasses training the RL algorithm through synthetic data. Two discrete synthetic datasets were generated, each corresponding to the two distinct queue sizes—32 and 64 packets. The specific dataset input provided to the RL algorithm is contingent upon the actions undertaken by the agent. The agent is endowed with two distinct actions: either transitioning the queue size from 32 to 64 packets or vice versa. So, when the agent executes an action to effect a queue size alteration to 32 packets, the RL model is fed by synthetic data derived from the corresponding scenario. Conversely, when the agent executes a queue size modification to 64 packets, the RL model is furnished with synthetic data from the scenario predicated upon that specific queue size.
4. ***Perfomance Evaluation of RL Models:*** This phase is designed to stage a comprehensive evaluation of the two RL training paradigms: one executed within the confines of real setup and the other reliant on synthetic data engendered by the TimeGAN model. The principal aim here is not to ascertain optimal configuration parameters or enhanced performance for the RL model but to holistically gauge the degree of similarity exhibited by the outcomes in both scenarios. In essence, this evaluation seeks to determine whether the TimeGAN, as an alternative to real setups, can viably replicate the results obtained through RL training, thereby validating its potential to serve as a surrogate simulator without compromising the efficacy of RL training.

The continuous line illustrates an online component wherein the RL model undergoes real-time training (depicted in steps 1 and 3). Conversely, the GAN model undergoes training in an offline manner, as depicted in step 2. Concluding the sequence, step 4 embodies the comparative evaluation conducted between the RL model, trained dynamically in the real setup, and synthetic data engendered by the TimeGAN algorithm.

3.4.1.3. Dataset characterization

Our dataset, centered on the Video Application within the Programmable Data Plane context, comprises two distinct categories of metrics: video metrics and network metrics, each residing in separate datasets. The video metrics encompass FPS, bitrate, and buffer size. In contrast, the network metrics consist of queue depth at packet queuing (Enq Qdepth), packet queuing duration in microseconds (Deq Timedelta), and queue depth at packet dequeuing (Deq Qdepth).

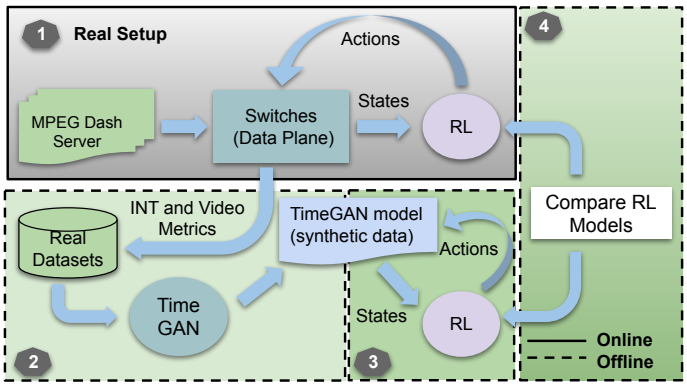


Figure 3.6: Proposed Methodology

Our network configuration employed BMv2 software switches with P4 code. We employed an out-of-band approach involving specific ONT probes sent from the DASH server to the Video Client. This approach eliminates the need to modify data packets for telemetry metadata. Telemetry metadata (32 bytes) was collected at each network node. Two experiments were conducted to gather data from our real setup, as discussed in Section 3.4.1.1. The first experiment used switches configured with 32-packet buffer size, while the second utilized a 64-packet buffer size. Consequently, we obtained two datasets, each representing the real setup under differing buffer size conditions. These datasets were merged based on timestamps, with higher 'Deq Timedelta' samples filtered out to capture network behavior under high-load conditions.

An essential characteristic of the telemetry values within our Programmable Data Plane application is the non-stationary nature of all our features, as visually demonstrated in Figure 3.7. This data distribution poses significant challenges in comparing real data with synthetic data generated by GANs. We introduce a metric in Section 3.4.1.5 to address this complexity. This metric serves as a tool in our model selection process, aiding us in identifying the most suitable model capable of providing the closest feature representation to the real dataset.

3.4.1.4. TimeGAN Training

TimeGAN is a generative model crafted to generate synthetic time series data. It intends to replicate the statistical characteristics inherent to real-world time series datasets. Training a TimeGAN model encompasses a series of critical steps.

To commence, we initiated the process with data preprocessing, addressing concerns such as missing data imputation, outlier removal, and data normalization. The configuration of hyperparameter values was also crucial since it plays a pivotal role in

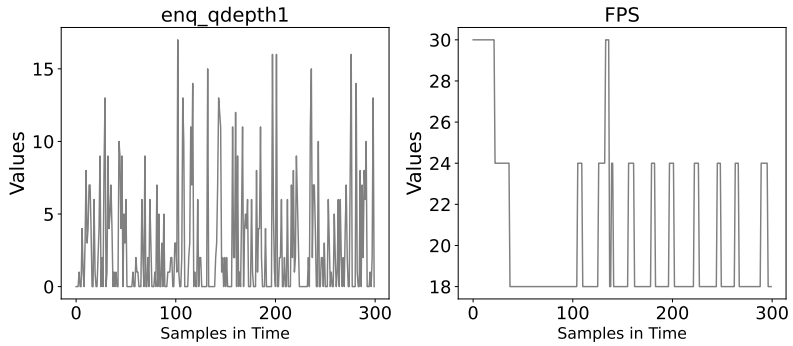


Figure 3.7: Real data plots for enq_qdepht1 and FPS features.

TimeGAN's training regimen. The determination of sequence sizes or windows was a significant consideration. These sizes are instrumental in the training, as they subdivide the dataset into multiple snapshots, each designed to capture the temporal behaviors within the time series.

Furthermore, other hyperparameters are variated, including sequence length, the number of hidden dimensions, and batch size. Hyperparameter tuning is a recognized challenge in machine learning training, with various strategies available, such as Grid Search. In our tutorial, we adopted an empirical approach, wherein we iteratively adjusted hyperparameter values based on prior training experiences and our insights into the parameters that exerted the most significant influence on the training process. After completing the training processes, we learned that the loss values across the training iterations exhibited a degree of similarity. Consequently, we recognized the need to develop a model selection methodology, the details of which are expounded upon in Section 3.4.1.5.

3.4.1.5. Models Selection

Addressing the challenge inherent to GAN, specifically the evaluation of synthetic data generated by these networks, is important. In [7] is underlined the absence of a consensus regarding the appropriate means to assess the distributions produced by GAN models. This issue is further compounded in the context of time series data, where the lack of recent publications is evident compared to more conventional GAN models. Besides, the characteristics of the dataset we collected are non-stationary. In contrast to stationary time series, non-stationary time series exhibit trends, seasonality, or other forms of systematic changes that make their statistical properties, such as the mean, variance, and autocovariance, vary with time. So, it is not possible to apply traditional tests such as Kullback–Leibler (KL) divergence. For example, the synthetic data obtained from certain trained models is constant values for some features, but real data has high variability. But, if a traditional test is applied, the results of the KL test are better for distributions with constant values, which is not desirable.

In our experimentation, we noted through empirical observations that certain GAN models we trained have superior synthetic data for distinct features. Consequently, we proposed a mechanism to select the most suitable model by analyzing each feature. To ad-

dress this objective, we introduce a metric constructed through straightforward statistical measures, encompassing quartiles and medians. The central concept revolves around contrasting the distribution disparities exhibited by real and generated synthetic data. Consequently, a more favorable model for a given characteristic is indicated by a diminished dissimilarity in data distribution between these two sources. The calculation of this metric is depicted by Equation 1. To initiate this calculation, it is imperative to determine the discrepancy magnitude between the third and first quartiles for both datasets: real (X) and synthetic (y). This calculation furnishes the size of data dispersion divergence, facilitating a comparison.

Subsequently, another calculation involves assessing the variation between the sizes of these data dispersion, thereby gauging the difference between real and synthetic data. However, determining the disparity in dispersion magnitude alone might be insufficient, as the dispersion might have similar sizes while maintaining a positional offset. To address this, we calculate the difference in medians between the real and synthetic datasets and incorporate this disparity alongside the variance in dispersion sizes. Thus, a summation of these differences between quartiles and median for each feature is achieved and stored in metric (M) for all models. So, a superior model is characterized by the minimal value of M, indicative of the least dispersion in data distribution between real and synthetic data sources.

$$\begin{aligned}\mathbf{M} = \sum_{n=1}^{n\_feats} & |[Q_3(X_n) - Q_1(X_n)] - [Q_3(y_n) - Q_1(y_n)]| \\ & + |[med(X_n) - med(y_n)]|\end{aligned} \tag{1}$$

The algorithm presented in Algorithm 1 elucidates the procedure of selecting the optimal model for individual features. The algorithm takes as its inputs arrays of models about the two distinct scenarios expounded upon in Section 3.4.1.4, specifically those delineated by switch queue sizes of 32 and 64 packets.

Algorithm 1 Optimal Model Selection

Input: $models32, models64$
Output: $bestModel32, bestModel64$
Initialisation :
1: $modelsMetrics32 \leftarrow calculateMetricForEachFeature(models32)$
2: $modelsMetrics64 \leftarrow calculateMetricForEachFeature(models64)$
3: **for** each $model32$, $model64$ in $modelsMetric32$ and $modelsMetric64$ **do**
4: $models32Sum[model] \leftarrow getSumFeaturesMetrics(model, modelsMetrics32)$
5: $models64Sum[model] \leftarrow getSumFeaturesMetrics(model, modelsMetrics64)$
6: **end for**
7: $bestModel32 \leftarrow min(models32Sum)$
8: $bestModel64 \leftarrow min(models64Sum)$
9: **return** $bestModel32, bestModel64$

As detailed earlier, the algorithm's initial operation involves the computation of the aforementioned metrics for each feature and all of the trained models. After this metric computation, an iteration wherein the algorithm determines the summation of metrics for each model is achieved. So, the minimum value of the sum of all features determines the best model ***considering the models trained in our study***.

The algorithm culminates by providing two arrays, each comprising the features of the best model. These arrays distinctly encapsulate the finest model under the respective queue size scenarios, addressing the dual scenario of queue sizes - 32 and 64 packets.

An evaluation of the model selection outlined in this section is expounded upon in Section 3.5.1.5. The primary objective is to provide a comprehensive analysis of the synthetic data generated by TimeGAN and its applicability in training an RL agent through an offline approach, contrasting it with an agent trained in an online format.

3.4.2. Generation of Synthetic Network Traces

This section discusses the NetDiffusion [25] and NetShare [51] problem definition and methodology towards synthetic packet generation.

3.4.2.1. NetDiffusion: Problem Definition

The NetDiffusion's authors proposed a two-fold strategy that leverages stable diffusion techniques. First, they convert raw packet captures into image representations - i.e., with nPrint [22]. This process allows them to use powerful image-based techniques for further analysis. Then, they fine-tune a text-to-image diffusion model on these converted packet capture images to let the model learn the patterns and relationships within network traffic data. To ensure the generated packets resemble real network traffic, they employ controlled techniques and domain knowledge-based heuristics to maintain fidelity to real-world data and ensure semantic correctness.

3.4.2.2. NetDiffusion: Methodology

The aforementioned steps are broadly summarized in Figure 3.8. The approach is structured around three primary components:

1. Conversion of raw network traffic into image-based representations using nPrint.
2. Fine-tuning a Stable Diffusion model to enable controlled text-to-traffic generation with high distributional similarity to real-world network traffic.
3. Domain knowledge-based post-processing heuristics for detailed modification of generated network traffic to ensure high protocol rule compliance.

The authors represented the packets as images leveraging print representation and noticed network traffic data exhibits high dimensionality. For instance, fields are abundant between the IP and TCP headers alone (e.g., IP addresses, ports, sequence and acknowledgment numbers, flags, etc.). In summary, nPrint has a bit-level representation that takes accounting for all [3] potential header fields (even if not present in the original packet). However, there are some limitations. While this ensures a uniform input structure for ML

[3] "All" means all considered headers by the authors so far: IPv4, IPv6 (Fixed Header), TCP, UDP, ICMP, Payloads.

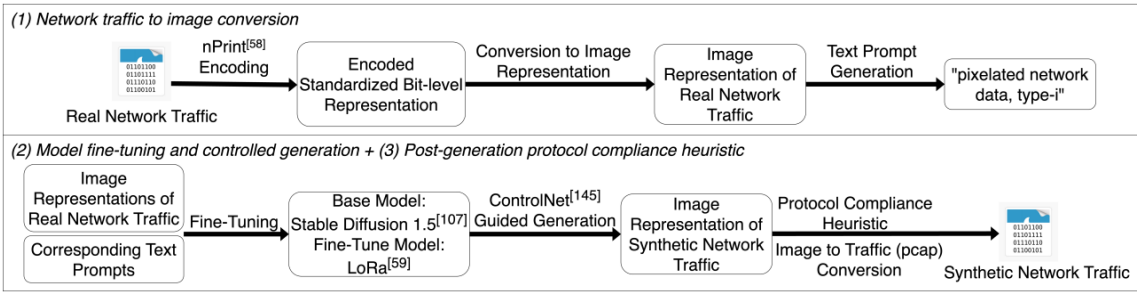


Figure 3.8: Generation Framework Overview. Source: the authors.

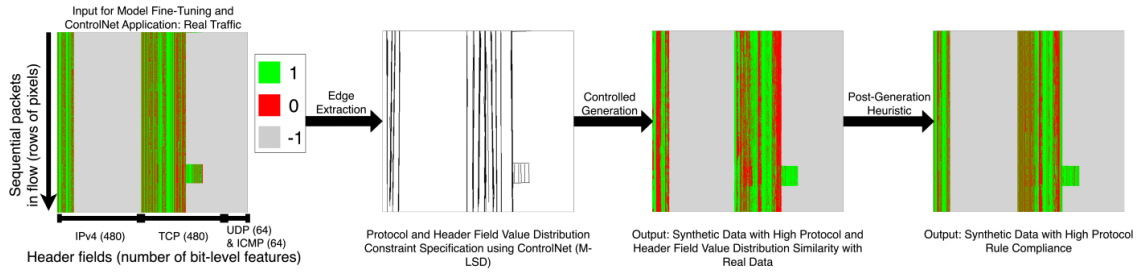


Figure 3.9: Synthetic Amazon network traffic outputs: (1) Using ControlNet, it detect regions present in the original traffic and ensure protocol and header field value distribution conformance by generating within specified regions. (2) Applying a post-generation heuristic to refine field details for protocol conformance. Source: The authors.

models, the attribute count per packet often exceeds a thousand. This high dimensionality introduces computational bottlenecks for generative models.

To arrive at image representations of network traffic, the authors first encode PCAPs using nPrint, which converts network traffic into standardized bits where each bit corresponds to a packet header field bit, as shown in Figure 3.8. This binary representation is simple yet effective, where the presence or absence of a bit in the packet header is denoted as 1 or 0, respectively, and a missing header bit is represented as -1. This encoding scheme ensures a standardized representation irrespective of the protocol in use. The payload content is not encoded since it is often encrypted. However, the size of the packet payloads can be inferred from other encoded header fields, such as the IP Total Length fields.

A sequence of PCAPs is converted into a matrix, which is then interpreted as an image. The colors green, red, and gray represent a set bit (1), an unset bit (0), and a vacant bit (-1), respectively. This color-coding schema provides a visually intuitive representation of the network traffic. Due to the limitations in the generative models' capability to handle very high dimensional data, they grouped every 1024 packets. While necessary for the current scope, this constraint could be revisited in future work to accommodate larger groups of packets. Through this process, any network traffic in PCAP format is transformed into an image with dimensions of 1088 pixels in width and 1024 pixels in height, with each row of pixels representing a packet in the network traffic flow as shown in Figure 3.9.

After transforming the packets into their corresponding image representations, the generative model is fine-tuned, specifically a diffusion model, to produce synthetic network traffic. While the out-of-the-box stable diffusion model is undeniably potent, they cannot directly use it to synthesize network traffic because it is designed to cover a broad spectrum of patterns and intricacies, causing it to lack the depth needed in specific generation tasks. For instance, a `Netflix Network Traffic` prompt might yield a generic image like a highway scene within a Netflix player. This is particularly evident when the textual description provided has multiple potential visual interpretations, causing the model to produce images that may be blurry or off-target. The authors addressed these limitations by fine-tuning Stable Diffusion on specific network datasets. They build upon Stable Diffusion 1.5 [43] and fine-tune this model on our specific network datasets as shown in Figure 3.8. To facilitate this fine-tuning, they employ Low-Rank Adaptation (LoRa) [23], a training technique tailored to fine-tune diffusion models, particularly in text-to-image diffusion models, swiftly. Its crux lies in enabling the diffusion model to learn new concepts or styles effectively while maintaining a manageable model file size. With LoRa, the resultant models are compact, balancing file size and training capability, while adpating to new data. For each image, they craft a unique encoded text prompt (e.g., `pixelated network data, type-0`) for Netflix traffic that succinctly describes its class type. The choice of their encoded prompt, though seemingly simplistic, achieves two main objectives. It offers a specific vocabulary that reduces ambiguity and ensures the model hones in on the nuances of the traffic. Subsequently, image-text pairs are fed into the fine-tuning process, where the base stable diffusion model, augmented with LoRa, learns to generate network traffic images conditioned on our prompts.

Upon fine-tuning the generation model, the next phase involves generating the desired class of synthetic network traffic. This is achieved by supplying the appropriate text prompts to the diffusion models to produce image representations of the traffic. Diffusion models operate by simulating a reverse process from a simple noise distribution to the data distribution, which enables them to capture and replicate the intricate patterns inherent in real-world data. The noise is progressively reduced over several steps, allowing the model to gradually refine the generated image until it closely resembles genuine network traffic patterns. An example of synthetic network traffic is shown in the image representation for Amazon traffic in Figure 3.9. This prompt-based generation process facilitates the creation of a synthetic nPrint-encoded network traffic dataset tailored to specific class distribution requirements. For instance, to curate a dataset with a certain class distribution and size, one would provide the corresponding quantity of text prompts per class and activate the generation process accordingly. Certain constraints are introduced during the generation process to ensure the generated traffic closely aligns with the prevalent protocol and header field value distributions observed in real traffic. If, for instance, the actual Amazon network traffic primarily consists of TCP packets, the generation process should prioritize populating header fields associated with TCP packets.

Leveraging the controllable nature of diffusion models, they incorporate ControlNet [55] into the generation process. ControlNet is a commonly used neural network architecture that adds spatial conditioning controls to large, pre-trained text-to-image diffusion models. It capitalizes on the robust encoding layers of these models, which are pre-trained with vast datasets, to learn a diverse set of conditional controls. In their spe-

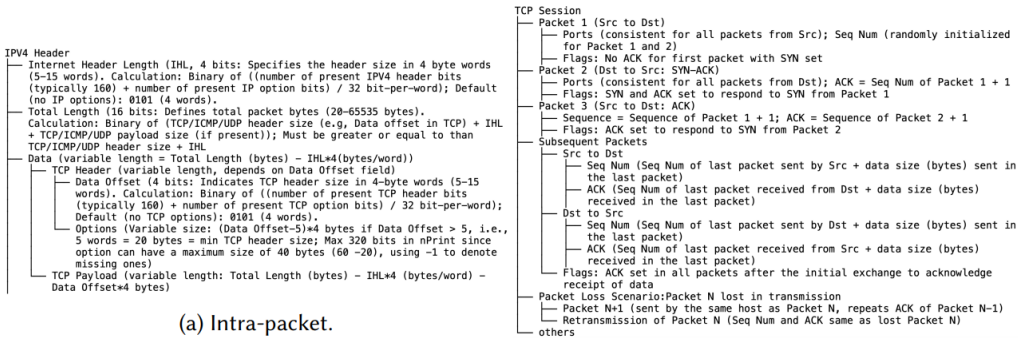
cific use case of ControlNet, they leveraged M-LSD straight-line detection to find the boundaries between fields that are supposed to be populated and those that are not, as shown in Figure 3.9. Other edge detection methods, such as Canny edge detection, produce similar results. Such line or edge detection methods are effective because they align with the inherent columnar consistency present in packet traces.

Utilizing the controlled diffusion model, the generated encoded network traffic resembles the protocol and header field value distributions inherent in real-world data, capturing every feature observed in real traffic. Despite the guidance provided by ControlNet during the generation process, some network analysis and testing tasks often require raw network traffic. Converting the synthetic encoded traffic back to raw formats, such as PCAP, is not straightforward. This complexity arises from the multitude of detailed transport and network layer protocol rules at both inter and intra-packet levels. Properly formatted traffic must strictly adhere to these rules. While transport layer rules emphasize end-to-end communication and reliability, network layer protocols focus on packet routing and address assignment. Combined, these rules can be broadly divided into two categories:

1. *Inter-Packet Rules:* These rules dictate the relationships and sequencing between header fields within multiple packets in a network flow. For instance, packets need to be sequenced properly in a typical TCP connection, starting with the handshake process involving SYN and SYN-ACK flags. Data transfer integrity is ensured by aligning sequence numbers and acknowledgment numbers. Misalignment or incorrect sequencing can disrupt the connection or data transfer process.
2. *Intra-Packet Rules:* These pertain to the structure and contents within individual packets. For example, many protocol headers have a checksum field computed based on the packet's contents to detect errors during transmission. The checksum must be consistent with the packet's payload. Additionally, certain fields within a packet, such as port numbers in TCP and UDP headers, must adhere to specific formatting and value constraints to ensure the packet's validity and proper routing.

To maximize the encoded synthetic network traffic's compliance with transport and network layer protocol rules, they first discern a subset of critical header fields that mandate strict adherence to their formatting rules, e.g., sequence and acknowledgment numbers. In contrast, some fields can accommodate a degree of flexibility without compromising the integrity of the network traffic, such as TCP window size or TTL With the critical fields identified, they develop a systematic way to calculate their correct values based on other generated fields. This is achieved by constructing two dependency trees — intra-packet header field dependencies and inter-packet dependencies. These trees are built upon domain knowledge and are sourced from standard network protocol documentation [6]. They present example protocol rules and the associated dependency trees for TCP protocol in Figure 3.10. More comprehensive and detailed dependency trees can be found in the open-sourced repository. [4] Given a generated encoded network traffic, they begin the correction process by traversing the trees in an automated, bottom-up fashion.

[4]`https://github.com/noise-lab/NetDiffusion_Generator`



(a) Intra-packet.

(b) Inter-packet.

Figure 3.10: Example TCP protocol rules/dependencies. Source: the authors.

Initially, they satisfy intra-packet dependencies, ensuring that individual packets are internally consistent. Subsequently, they address inter-packet dependencies, guaranteeing that the packets in a flow relate correctly. Certain fields necessitate uniformity across packets within the same network traffic trace—like IP addresses and ports. Others require specific initialization values, such as IP identification and TCP acknowledgment numbers. They employ a majority voting system to determine the most appropriate values for these fields by selecting the most frequently appearing value within the generated traffic. Another notable challenge is timestamp assignment for individual packets, given its intricate time series dependencies. This post-processing ensures that the encoded synthetic traffic can be seamlessly converted into raw network traffic formats (like PCAP) and subsequently be utilized for a range of non-ML tasks.

3.4.2.3. NetShare: Problem Definition

In this work, the authors explored the feasibility of ML-based synthetic packet-header (e.g., PCAP) and flow-header (e.g., Netflow) trace generation using Generative Adversarial Networks or GANs. They implement an end-to-end system, NetShare, and build a web service prototype [5] to help solve the following practical challenges in existing approaches:

- *Prior techniques*: (especially those based on tabular data GANs, which dominate the synthetic header generation literature) are unable to capture key correlations across header fields and header fields that have large ranges of values.
- *Scalability-fidelity tradeoff:* Existing techniques require significant GPU hours to train even moderately-sized traces (e.g., millions of records). Simple tabular GANs take a few hours to train but suffer in fidelity, while more complex time series GANs can take an order of magnitude more time.
- *Privacy-fidelity tradeoffs:* GANs are not well explored in the context of network header traces. Preliminary work suggests that differentially-private learning approaches are likely to yield poor fidelity for networking datasets [28].

[5] Available through `https://www.pcapshare.com`

In designing NetShare, the authors tackle these key challenges with a careful data-driven understanding of the limitations of canonical GAN-based approaches. NetShare combines the following key ideas to address the above issues:

- *Reformulation as flow time series generation:* Instead of treating header traces from each measurement epoch as an independent tabular dataset (i.e., rows of packets/flows with headers), they recast the problem for learning synthetic models for a merged flow-level trace across epochs. This reformulation allows us to capture intra and inter-epoch correlations natively.

- *Improving scalability via fine-tuning :* They identify opportunities to optimize learning time by using ideas of model fine-tuning and data-parallel learning from the ML literature [53]. Doing so naively may fail to capture dependencies across parallel instances, so they develop heuristics to preserve such correlations.

- *Practical privacy reformulations:* They adopt recent advances in differentially-private model training [54] and combine a small amount of public data with private data to improve privacy fidelity tradeoffs. To the best of our knowledge, this is the first application and empirical demonstration of header trace generation.

3.4.2.4. NetShare: Methodology

NetShare [6] is an end-to-end system leveraging a tabular representation distributed among different GANs with temporal and confinement dependencies. According to the authors, the existing approaches treat each packet or flow record independently and ignore intra- and inter-measurement epoch correlations. To systematically capture these cross-record correlations, they reformulate the header generation problem as a time series generation problem rather than a tabular generation problem, as shown in Figure 3.11. Specifically, they merge data from measurement epochs into one giant trace to capture inter-measurement epoch correlations. Given this giant trace, they split it into a set of flows. D^{flow} based on five-tuples to explicitly capture flow-level metrics (e.g., flow size/duration). Each sample in D^{flow} has a five-tuple as metadata and a record (or "measurement data") corresponding to a sequence of packets for PCAP data and flow records for NetFlow data. Specifically, for PCAP data, each sequence element (packet) includes a raw timestamp, packet size, and other IP header fields.

[6]The code is open-sourced at `https://github.com/netsharecmu/NetShare`.

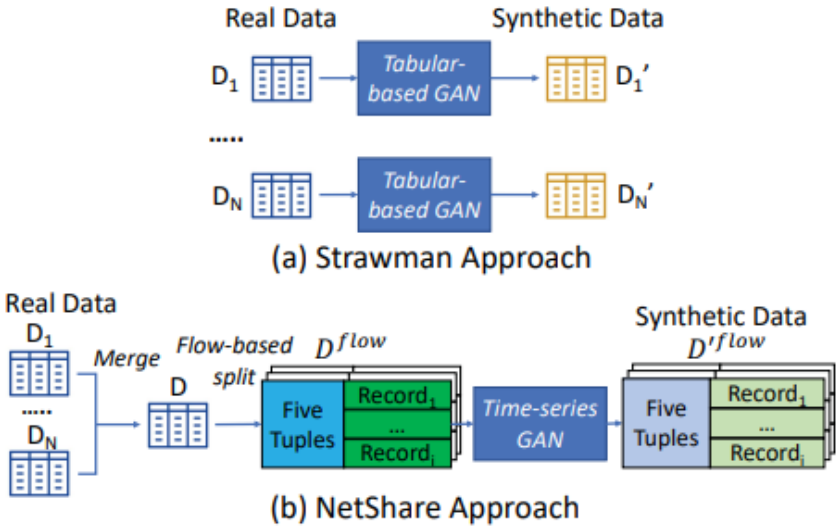


Figure 3.11: Multiple epochs D_i merged into a giant trace D, splitting the trace into flows D^{flow}, and using time series GAN. Source: the authors.

Reformulating the problem as a time-series generation brings much better header/temporal correlations but increases the total CPU time. One opportunity to improve scalability is via parallelism. However, naively dividing the giant trace into chunks and parallelized training across chunks poses two limitations. First, they incur the risk of losing correlations across chunks [7] e.g., flow size distribution for flows that span multiple chunks. Second, while the wall clock time decreases, the total CPU hours consumed remain unchanged.

These limitations can be avoided as shown in Fig 3.12. First, they borrow the idea of fine-tuning from the ML literature, i.e., they use a pre-trained model as a "warm start" to seed training for future models [54]. Specifically, they use the first chunk as the "seed" chunk to give a warm start, and subsequent chunks are fine-tuned using the model trained from the first chunk. This permits parallel training across chunks. One concern remains regarding the cross-chunk correlations; fine-tuning alone cannot preserve these. To this end, they append "flow tags" to each flow header to capture the inter-chunk correlation. Specifically, they annotate each flow header with a 0-1 flag denoting whether it starts in "this" chunk. They append a 0-1 vector after the flag with a length equal to the total number of chunks, with each bit indicating whether the flow header appears in that specific chunk. When splitting the giant trace D^{flow} flow into chunks, there are two natural choices: split by fixed time interval or by number of packets per chunk. Splitting by a fixed number of packets per chunk may impact differential privacy guarantees, as any single packet could change the final trained model in an unbounded way: removing any packet could change the packet assignment of all the following trunks. Thus, they split by fixed time intervals rather than a fixed number of packets. They left the choice of M (number of chunks) as a configurable tradeoff; a higher M would give fewer total CPU hours while increasing the learning complexity across chunks. In their case, $M = 10$ was

[7] These chunks are logically independent of the measurement epochs in the original dataset; chunks are merely a construct for parallelization training.

chosen for each dataset with 1 million records.

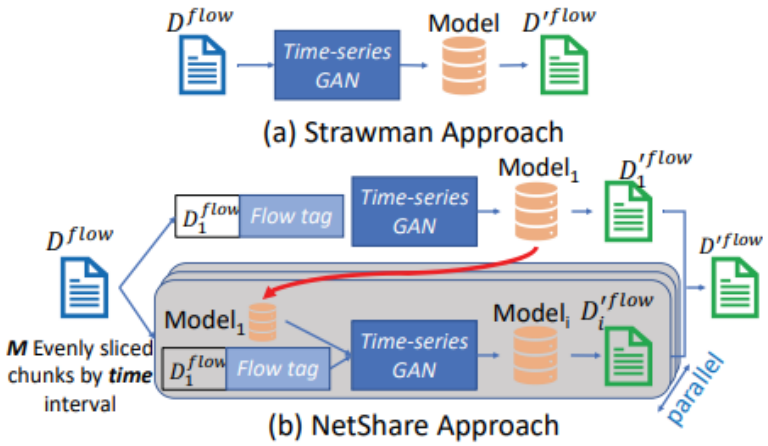


Figure 3.12: They split D^{flow} into M evenly time-spaced chunks with explicit "flow tags" to capture cross-chunk correlations. They use the first chunk as a pre-trained model for parallel training of later chunks. Source: the authors.

Prior attempts to train DP synthetic network data models using deep generative models have utilized DP-SGD, which modifies stochastic gradient descent (SGD) by clipping each gradient and adding Gaussian noise [1]. For a fixed amount of added noise, the more rounds of DP-SGD they run, the greater the cost in the privacy budget. In NetShare, they exploit the observation that one can reduce the number of rounds of DP-SGD needed to achieve a fixed fidelity level by pre-training NetShare on a related public dataset; then, they take the learned parameters from the public dataset and fine-tune them using DP-SGD over the private dataset. In doing so, they reduce the required number of iterations of DP-SGD. This insight has been explored in related work from the DP community [5, 26], but it has not been utilized in the networking domain to the best of our knowledge. They also use public data to improve our privacy-fidelity tradeoff due to our IP2Vec encoding. Specifically, they train our IP2Vec mapping on a public dataset with many port/protocol pairs, which helps us learn an embedding without affecting their DP budget.

Finally, they use a time series GAN to model this data. While autoregressive models [50] also use a time series approach, they are less effective for learning implicit distributions (e.g., flow length [28]) and achieve worse fidelity. Specifically, they build on an open-source tool called DoppelGANger [28]. Note, however, that natively using a time-series GAN like DoppelGANger would run into the same issues as the tabular GANs, as each flow or packet record will be a time-series record of length 1 and will miss the key cross-record effects. Furthermore, they will also encounter other challenges regarding encoding, scalability, and privacy. The baselines struggle to learn the distribution of fields with large support accurately. Hence, instead of training a GAN on the original data representation, they use domain knowledge to transform certain fields (especially those with large support) into a more tractable format for GANs. For fields with numerical semantics like packets/bytes per flow with a large support, they use log transformation, i.e., $\log(1+x)$, to effectively reduce the range. This simple yet effective technique helps NetShare better distribute large-support fields than baselines.

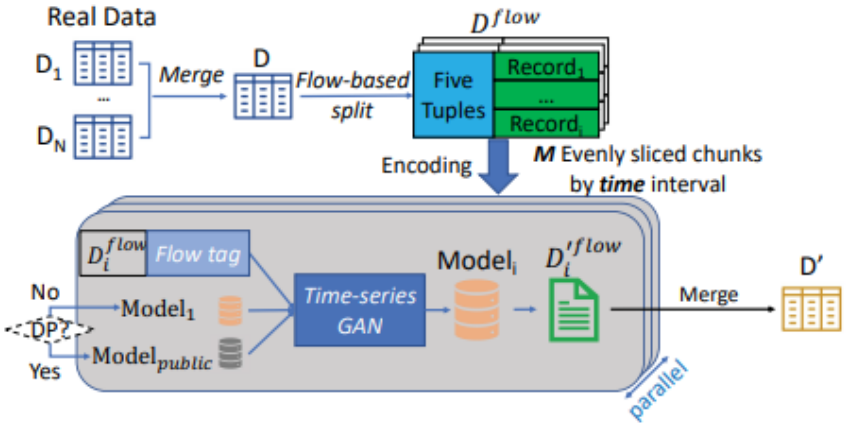


Figure 3.13: NetShare: end-to-end overview. Source: the authors.

3.4.2.5. NetShare: End-to-end view

Combining the key insights above, our end-to-end design is summarized in Figure 3.13. **Pre-processing:** Merge data from different measurement epochs D_i into one giant trace D with a flow-based split as D^{flow}. Encode header fields based on domain knowledge and fidelity-scalability-privacy tradeoffs.

Training: Evenly slice flow traces into M fixed-time chunks with explicit flow tags added. Train a time series GAN for each chunk; they use DoppelGANger [28]. If differentially-private (DP) is not required, use the model from the first chunk as the pre-trained model to improve scalability-fidelity tradeoff. If DP training is desired, use the model pre-trained on public data to fine-tune DP-SGD.

Post-processing: After generating $D_i' flow$, they map transformed fields back to their natural representations (e.g., map IP2Vec embeddings to `<port, protocol>` via nearest-neighbor search). Then, they generate derived fields (e.g., checksum) [8]. Finally, they convert to PCAP/NetFlow dataset by merging packets/NetFlow records according to the raw timestamp (for PCAP) or raw flow start time (for NetFlow).

3.5. Use Cases for Synthetic Data Application

This section presents the practical application of the two use cases outlined in Section 3.1. We aim to demonstrate the methodologies, code snippets, and tools for creating synthetic datasets. Specifically, Section 3.5.1 explores the generation of telemetry data for INT and DASH protocols, detailing the process and techniques involved. Conversely, Section 3.5.2.1 focuses on creating PCAP files, elucidating the procedure using the NetDiffusion Tool.

[8]They make an explicit design choice to exclude such derived fields, which are likely intractable to learn automatically. As such, they use a two-step generation mechanism: (1) use NetShare to generate the native fields (e.g., IP/port/timestamp) and (2) compute the checksum based on that to ensure the correctness of packets. Additionally, they did not consider the option field in the IP header, which is rarely used (and they do not observe the appearance of the IP option field in all three PCAP-related datasets).

3.5.1. Generation of Telemetry Data

This section guides training, generating, evaluating, and applying synthetic telemetry data for training an RL agent. It's important to note that this chapter does not cover the initial phase of our methodology, which involves data collection in a real-world environment. Our primary focus here is on the synthesis of data. The entire project for this tutorial can be accessed and cloned from the Github repository [9]. We have used in all of our scripts the YData Synthetic API[10], to conduct and perform the training and data generation.

The creation, training, and evaluation of synthetic data are heavily influenced by the number of models you aim to develop, each featuring a unique set of hyperparameters. To streamline this process, we provide a configuration file (**`params.py`**), which defines essential parameters for our workflow. In this tutorial, we introduce a variable called `amount_of_models` that influences the setting of hyper-parameter values, including:

- **`seq_len` (sequence length)**: This parameter delineates the temporal window's extent for each sequence during the training phase, meaning the count of time chunks (or lines) encapsulated within each sequence.
- **`hidden_dim` (hidden dimensions)**: It specifies the count of units or neurons present within each hidden layer of the network, thus shaping the model's capacity to learn and represent data complexities.
- **`batch_size`**: This hyperparameter defines the quantum of temporal sequences (or the number of data examples/lines) that are congregated into a singular batch for training, affecting both the computational efficiency and the learning dynamics of the model.
- **`train_steps`**: It denotes the aggregate count of training iterations, quantifying the extent of the model's exposure to the training data, influencing its learning trajectory and convergence behavior.

The variable `amount_of_models` is factored to set the upper limits for the `i`, `j`, and `k` variables in the scripts' nested triple-loop structure. The `fatNum` method in `ModelUtility` class is designed to compute these maximum values. For instance, with three models, the maximum limits for `i`, `j`, and `k` are set to `1`, `1`, and `3`, respectively. With nine models, these limits are adjusted to `1`, `3`, and `3`, respectively.

So, the workflow for synthetic data generation is systematically outlined across several sections. Section 3.5.1.1 describes the environment configuration. Section 3.5.1.2 details the data preprocessing steps, Section 3.5.1.3 focuses on the training process, Section 3.5.1.4 describes the synthetic data generation, and Section 3.5.1.5 examines the evaluation of the generated synthetic data. Each section is dedicated to a specific workflow phase, ensuring an understanding of the entire synthetic data generation process.

[9] https://github.com/thiagocaproni/tutorial_timegan

[10] https://github.com/ydataai/ydata-synthetic.

3.5.1.1. Dependencies and Environment Settings

Python 3.9.16 is necessary to execute the code. An environment manager like Anaconda, which includes the conda package manager, can be used to set up a dedicated environment with all the required dependencies. Installing Anaconda in your user directory ensures it does not conflict with the existing system Python installation. The dependencies are listed in the **environment.yml** file.

To create and configure the environment, execute the command provided in Code 3.1 within the repository directory, where **environment.yml** file is located. This will establish a separate environment and install the necessary dependencies. To begin using this environment, activate it by executing the command "conda activate".

Code 3.1: Environment Settings

```
conda env create
conda activate ydata
```

3.5.1.2. Data preprocessing

The `DataPre` class (defined in **`preprocess_data.py`**) is a component in our data preprocessing pipeline, designed specifically for preparing time series data for synthetic data generation. In this part, we will delve into each method provided by this class and explain how they contribute to the preprocessing of the dataset. Before we start utilizing it, we need to import the necessary Python libraries that our class depends upon, as demonstrated in the script in the repository.

The `transformTimeStamp` method of `DataPre`, shown in Code 3.2, is used for standardizing the timestamp format across our datasets. This method converts timestamps from milliseconds to seconds and sets them as the DataFrame's index, a common prerequisite for time series analysis.

Code 3.2: Transforming Timestamps

```
def transformTimeStamp(self, df):
    df['timestamp'] = df['timestamp'] / 1000
    df['timestamp'] = df['timestamp'].astype(int)
    df.set_index('timestamp', inplace=True)
```

In turn, Code 3.3 presents the `loadDataSet` method, which is responsible for loading and combining multiple data sets into a single DataFrame. This method ensures that data from different sources can be aligned and analyzed based on timestamps.

Code 3.3: Loading and Merging Datasets

```
def loadDataSet(self, path_int, path_dash):
    df_int = pd.read_csv(path_int, sep=',')
    df_dash = pd.read_csv(path_dash, sep=';')
    self.transformTimeStamp(df_dash)
    df_int = df_int.loc[df_int.groupby('timestamp')['deq_timedelta1'].idxmax()]
    df_int.set_index('timestamp', inplace=True)
    self.dataset = pd.merge(df_int, df_dash, left_index=True, right_index=True).
    reset_index()
```

The `hotEncode` method (Code 3.4) applies one-hot encoding to categorical variables in the dataset, transforming them into a format that machine learning algorithms can more effectively utilize.

Code 3.4: One-Hot Encoding of Categorical Variables

```
def hotEncode(self):
    self.encoder = OneHotEncoder(handle_unknown='ignore')
    encoder_df = pd.DataFrame(self.encoder.fit_transform(self.dataset[['Resolution']]).
    toarray())
    self.dataset = self.dataset.join(encoder_df).copy()
    self.dataset.drop('Resolution', axis=1, inplace=True)
```

Finally, the `preProcessData` method orchestrates the overall preprocessing workflow, covering the imputation of missing values, encoding, and optional data shuffling.

Code 3.5: Preprocessing the Dataset

```
def preProcessData(self, num_cols, cat_cols, random):
    for i in num_cols:
        self.dataset[i].fillna(self.dataset[i].mean(), inplace=True)
    if len(cat_cols) > 0:
        self.hotEncode()
    self.processed_data = self.dataset[num_cols + cat_cols].copy()
    if random:
        idx = np.random.permutation(self.processed_data.index)
        self.processed_data = self.processed_data.reindex(idx)
```

The `DataPre` class serves as a tool for preparing time series data for synthetic data generation, encompassing a variety of preprocessing tasks to ensure data quality and compatibility for model training, which is discussed in Section 3.5.1.3.

3.5.1.3. TimeGAN Training

In practice, the scripts `timegan32.py` and `timegan64.py` perform data preprocessing and train a TimeGAN model for the synthetic generation of time series data. In this section, we show the code step by step, explaining each method's purpose and how they collectively form a pipeline for generating synthetic data. As in Section 3.5.1.2, we start by importing (see the script in the repository) the necessary Python modules and libraries and adjusting the system path to include our utility scripts.

The `loadDp` method, presented in Code 3.6, orchestrates the data loading and preprocessing, leveraging the `DataPre` class for various preprocessing tasks.

Code 3.6: Loading and preprocessing data

```
def loadDp(random, outliers):
    dp = DataPre()
    dp.loadDataSet(path_int='../../datasets/log_INT_TD-32_100.csv',
                   path_dash='../../datasets/dash_TD-32_100.csv')
    dp.preProcessData(params.num_cols, cat_cols=params.cat_cols, random=random)
    dp.removeSameValueAttributes()
    if not outliers:
        dp.removeOutliers()
    return dp
```

The `train` function, as shown in Code Listing 3.7, produces the setup and execution of the TimeGAN model's training process. This method utilizes the preprocessed data alongside the TimeGAN instance, which is constructed using the YData API, to train the model effectively.

Code 3.7: Training a model

```
def train(dp, seq_len, n_seq, hidden_dim, noise_dim, dim, batch_size, model, train_steps
    ):
    learning_rate = 5e-4
    gan_args = ModelParameters(batch_size=batch_size, lr=learning_rate, noise_dim=
    noise_dim, layers_dim=dim)
    processed_data = real_data_loading(dp.processed_data.values, seq_len=seq_len)
    synth = TimeGAN(model_parameters=gan_args, hidden_dim=hidden_dim, seq_len=seq_len,
    n_seq=n_seq, gamma=1)
    synth.train(processed_data, train_steps=train_steps)
    synth.save(model)
```

The scripts `timegan32.py` and `timegan64.py` initiate the data preprocessing and model training processes, iterating over various hyperparameter configurations to optimize model performance. In Code 3.8, we illustrate a portion common to both scripts, focusing on the model training procedure. This code part demonstrates a triple-nested loop structure, where each iteration level adjusts specific hyperparameters critical to the training dynamics. As previously discussed, this adjustment of hyperparameters is important for refining the model's ability to effectively generate realistic synthetic time series data.

Code 3.8: Executing the training process

```
dp = loadDp(random=False, outliers=False)
iMax, jMax, kMax = ModelUtility.fatNum(params.amount_of_models)
print("Number of models", params.amount_of_models, 'iMax:', iMax, 'jMax:', jMax, 'kMax:'
    , kMax)
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
try:
    with tf.device('/device:GPU:0'):
        for i in range(iMax):
            for j in range(jMax):
                for k in range(kMax):
                    train(dp, ...)
except RuntimeError as e:
    print(e)
```

So, this Section explored the complete loading, preprocessing, and training of a TimeGAN model for synthetic time series data generation. The workflow cohesively integrates data handling and machine learning training, preparing the data and training the model with varying configurations to generate high-quality synthetic time series.

3.5.1.4. Telemetry Data Generation

This section outlines the process for creating synthetic time series data with TimeGAN. It includes steps for loading and preprocessing real datasets and performing statistical analysis. These activities form the basis for assessing the quality of the synthetic data, which will be further explored in the evaluation script detailed in Section 3.5.1.5. The synthetic data generation is carried out using the script `generate_synth_data.py` available in our Git repository.

The first thing to do is import the required libraries and modules (see the repository). Adjust the system path to include directories where additional modules like `DataPre` (discussed in Section 3.5.1.2) and `ModelUtility` are located.

The `loadSynthData` method, presented in Code 3.9, is designed to generate synthetic data based on previously trained TimeGAN models. It loads the models and generates a specified number of synthetic data windows. This method generates synthetic data based on models trained in the **timegan32.py** and **timegan64.py** scripts. The object returned when generating synthetic data is a three-dimensional object (`[i][j][k]`), where the variable `i` controls the number of windows generated and defined by the variable `seq_len` (explained in the training script). In turn, the variable `j` controls the index of the lines in each window, and the variable `k` controls the columns. In other words, several time windows are returned. Therefore, it is necessary to define the number of windows to generate in the `loadSynthData` method.

Code 3.9: Loading and creating synthetic data

```
def loadSynthData(model32, model64, number_of_windows):
    # Load TimeGAN models for 32-bit and 64-bit data
    synth_32 = TimeGAN.load(model32)
    synth_data_32 = synth_32.sample(number_of_windows)
    synth_64 = TimeGAN.load(model64)
    synth_data_64 = synth_64.sample(number_of_windows)
    # Post-process synthetic data (e.g., binarizing a portion of the data, which was hot
     encoded)
    synth_data_32[:,:,13:17][synth_data_32[:,:,13:17] >= 0.5] = 1
    synth_data_32[:,:,13:17][synth_data_32[:,:,13:17] < 0.5] = 0
    synth_data_64[:,:,13:17][synth_data_64[:,:,13:17] >= 0.5] = 1
    synth_data_64[:,:,13:17][synth_data_64[:,:,13:17] < 0.5] = 0
    return synth_data_32, synth_data_64
```

The `loadRealData` function is responsible for loading and preprocessing the real datasets for subsequent comparison with synthetic data.

Code 3.10: Reading real data

```
def loadRealData(dsint32, dsint64, dsdash32, dsdash64, num_cols, cat_cols, sample_size,
    randon, outliers):
    # Instantiate DataPre for each dataset and process
    dp32 = DataPre()
    dp32.loadDataSet(path_int=dsint32, path_dash=dsdash32)
    dp32.preProcessData(num_cols, cat_cols=cat_cols, random=randon)
    if not outliers:
        dp32.removeOutliers()
    real_data_32 = dp32.processed_data
    real_data_32 = real_data_32[0:sample_size].copy()
    real_data_32 = real_data_32.values
    # Repeat for 64-bit data
    dp64 = DataPre()
    dp64.loadDataSet(path_int=dsint64, path_dash=dsdash64)
    dp64.preProcessData(num_cols, cat_cols=cat_cols, random=randon)
    if not outliers:
        dp64.removeOutliers()
    real_data_64 = dp64.processed_data
    real_data_64 = real_data_64[0:sample_size].copy()
    real_data_64 = real_data_64.values
    return real_data_32, real_data_64
```

Calculate statistical metrics such as quartiles and median to assess the quality of the synthetic data, as demonstrated in Code 3.11. It is important to highlight that the `genStatisctics` method returns a dictionary for a given set of real and synthetic

models comprising lists of column statistics (features). Each list contains statistical values in positions 0, 1, 2, and 3, corresponding to real data from 32-bit real, 32-bit synthetic, 64-bit real, and 64-bit synthetic, respectively, presenting percentiles and medians.

Code 3.11: Statistical methods

```
def getStatistics(data):
    # Compute and return percentiles and median
    median = np.median(data)
    percentile_25 = np.percentile(data, 25)
    percentile_75 = np.percentile(data, 75)
    return [percentile_25, median, percentile_75]

def genStatisctics(real_32, synth_32, real_64, synth_64, sample_size, num_cols):
    # Generate statistics for each column in the dataset
    statistics = {}
    for col in num_cols:
        statistics[col] = [getStatistics(real_32[:, col][:sample_size]),
                           getStatistics(synth_32[:, col][:sample_size]),
                           getStatistics(real_64[:, col][:sample_size]),
                           getStatistics(synth_64[:, col][:sample_size])]
    return statistics
```

So, the Code 3.12 is defined to convert the three-dimensional synthetic data objects into a two-dimensional dataset, flattening the windows into rows and columns.

Code 3.12: Creating dataset in two dimensions

```
def createDataSet(seq_len, data):
    lines = int(params.synth_sample_size / seq_len)
    dataset = np.zeros(lines * seq_len * params.merged_columns_len).reshape(lines *
    seq_len, params.merged_columns_len)
    for i in range(lines):
        for j in range(seq_len):
            dataset[i * seq_len + j] = data[i][j][:]
    return dataset
```

Code 3.13, especially in `get_allfeatures_metrics`, presents the metric generation based on Equation 1, which is derived from the discrepancy in the interquartile ranges between each feature of the real and synthetic data. Additionally, the variance in the medians of these features is incorporated into this computation. Consequently, an object named 'metrics' is instantiated, encapsulating the results of this equation across all models and their corresponding characteristics. Subsequently, these metrics are used in the evaluation script to determine the optimal model. It should be emphasized that the total of the metrics for all features of a given model, denoted by the variable M in the equation, is calculated in the `analyze_data_models.ipynb` script in the method `getFeaturesBestMetricsOfModels`. This method identifies the best and worst models from the trained models.

Code 3.13: Method of creating metrics

```
def getMetrics(statistic_data):
    # Calculate discrepancy metrics for 32 and 64-bit data
    metric32 = abs(statistic_data[0][2] - statistic_data[0][0]) - abs(statistic_data
    [1][2] - statistic_data[1][0]) + abs(statistic_data[0][1] - statistic_data[1][1])
    metric64 = abs(statistic_data[2][2] - statistic_data[2][0]) - abs(statistic_data
    [3][2] - statistic_data[3][0]) + abs(statistic_data[2][1] - statistic_data[3][1])
    return metric32, metric64

def get_allfeatures_metrics(metrics, model_index, statistic_data):
    # Aggregate metrics for each feature across all models
```

```
    for col in params.num_cols:
        metrics[0][model_index][col], metrics[1][model_index][col] = getMetrics(
    statistic_data[col])
```

As detailed in Section 3.5.1.3 on the training script, the generation script employs a triple-nested loop to process all trained models, generating data and computing statistical metrics for each one. To maintain clarity in this document, we have opted not to include this specific code segment. For those interested in examining the full code, it is accessible in our Git repository.

This section methodically illustrated the process of generating synthetic time series data with TimeGAN. It covers everything from preparing real datasets and generating synthetic data to conducting statistical analyses to evaluate the synthetic data's quality. Section 3.5.1.5 delves into the script responsible for identifying the optimal model from all the trained models.

3.5.1.5. Model Selection

This section assesses synthetic time series data generated by TimeGAN, focusing on identifying the best and worst-performing models through statistical analysis. The Python notebook script, `analyze_data_models.ipynb`, available in our repository, contains the code for model selection. The initial step involves importing necessary Python libraries and setting up the environment.

So, The `getFeaturesBestMetricsOfModels` method in the script (Code 3.14) evaluates synthetic data generated by different models to identify the best and worst-performing models based on their statistical metrics. The method aims to compare the synthetic data generated by different models with the real data, identifying which models produce the most and least similar models to the real data based on the aggregated metrics. It starts by calling `sumFeatureMetricsOfModels` to get the aggregate metrics for each model. These metrics represent the total summation of the synthetic data from the real data across all features. Two lists are returned: `sum32` and `sum64`, respectively, representing the summed metrics for 32-bit and 64-bit model data. The function then identifies the best models (i.e., those with the lowest sum value in their metrics, indicating the least deviation from the real data) for both 32-bit and 64-bit data. It uses `np.argmin` to find the index of the lowest value in `sum32` and `sum64`, indicating the best models. The `getModelNameByIndex` method (see our git) retrieves the model name based on this index, and the model's synthetic data is accessed using this name.

Similarly, it identifies the worst models (those with the highest sum value in their metrics, indicating the greatest deviation from the real data) for both 32-bit and 64-bit data. It uses `np.argmax` to find the highest value index in `sum32` and `sum64` and retrieves the corresponding model's name and data. Finally, the function returns the synthetic data for the best and worst models for 32-bit and 64-bit buffer sizes.

Code 3.14: Evaluating the best and word models

```
def sumFeatureMetricsOfModels(models, data_metrics):
    # Initialize arrays to store sum of metrics for each model
    sum32 = np.zeros(len(models))
```

```
    sum64 = np.zeros(len(models))
    # Sum the metrics for each feature in 32 and 64-bit models
    for i in range(len(models)):
        sum32[i] = sum(data_metrics[0,i,:])
        sum64[i] = sum(data_metrics[1,i,:])
    return sum32, sum64

def getFeaturesBestMetricsOfModels(models, metrics):
    sum32, sum64 = sumFeatureMetricsOfModels(models, metrics)
    index = np.argmin(sum32)
    model = getModelNameByIndex(index)
    best_32 = models.get(model)[0]
    index = np.argmin(sum64)
    model = getModelNameByIndex(index)
    best_64 = models.get(model)[0]
    index = np.argmax(sum32)
    model = getModelNameByIndex(index)
    worst_32 = models.get(model)[0]
    index = np.argmax(sum64)
    model = getModelNameByIndex(index)
    worst_64 = models.get(model)[0]
    return best_32, worst_32, best_64, worst_64
```

The subsequent portion of the script, following the method to select the best and worst models, is focused on producing graphical representations to facilitate the analysis of the models. This includes generating box plots, violin plots, cumulative distribution function (CDF) plots, and correlation matrices. These visualizations are integral in understanding the performance nuances of the models identified by the selection method, highlighting the differences in data distribution, variability, and correlation patterns between the synthetic data generated by the best and worst models. Additionally, the script enables the exportation of the best model's dataset in a CSV format through `createSaveDataSetModel` method. This dataset can then be imported into the scripts designed for training the RL agent.

Also, it is important to be clear that in [52], various methods are outlined for assessing the quality of data produced by a TimeGAN model. These evaluation techniques include visualization tools like t-SNE and PCA, alongside regression models to analyze samples generated by TimeGAN models. The visualization method, however, necessitates human involvement for subjective assessment, making it challenging to determine the most appropriate model definitively. Conversely, the regression-based evaluation strategy was found to be less effective in our context due to the non-stationary nature of our data, complicating the task of developing a reliable regressor.

Despite these obstacles, we implemented a basic RNN via the Keras framework for regression analysis to compare real and synthetic data. Utilizing the Adam optimizer and Mean Absolute Error (MAE) as the loss metric, the RNN was configured to forecast the final entry in a sequence from its preceding elements. Nonetheless, the outcomes for both the real and synthetic datasets were unsatisfactory. The code for this regression model is accessible in our Git repository.

Due to the aforementioned issues, our approach to model selection was primarily grounded in statistical analysis. Moreover, we pursued an alternative method for evaluating our synthetic data by developing an RL agent, trained using both real and synthetic datasets, to evaluate its performance in each scenario, as detailed in Section 3.5.1.6.

3.5.1.6. Applying synthetic data to an RL agent

This section will present the implementation details regarding the queue management problem in video streaming applications described in section 3.4.1.1. We will start by describing how to model the agent-environment interaction illustrated in Figure 3.2. We will briefly explain how the agent learns, and lastly, we will show how to use the agent and environment classes in an application context.

In a video streaming application scenario, the `Environment` class will receive the synthetic INT metadata, execute an action, and calculate a reward value for such an action (Code 3.15). We modeled the agent-environment interaction by considering the DASH Server video streaming chunk size as a time step. Thus, the agent should take action to increase or decrease the switch's queue size every 4 seconds.

Code 3.15: Take action method from the Environment class

```
1  def take_action(self, action, intState, dashState):
2      self.current_state = intState
3      self.received_intStates.append(intState)
4      self.received_dashStates.append(dashState)
5      self.actions_history.append(action)
6      self.fps_history.append(dashState[FOURTH_SECOND][FPS_INDEX])
7
8      if action == BUFFER_SIZE_64:
9          self.buffer_size.append(64)
10     elif action == BUFFER_SIZE_32:
11         self.buffer_size.append(32)
12
13     if len(self.received_intStates) == 2:
14         # state observed when the action was taken
15         self.current_state = self.received_intStates[0]
16         # resulting state after the taken action
17         self.next_state = self.received_intStates[-1]
18         # dash state when the action was taken
19         current_dash = self.received_dashStates[0]
20         # dash state after the taken action
21         next_dash = self.received_dashStates[-1]
22         self.calculate_reward(current_dash[FOURTH_SECOND][BUFFER_INDEX], next_dash[
       FOURTH_SECOND][BUFFER_INDEX], next_dash[FOURTH_SECOND][FPS_INDEX])
23         self.received_intStates = []
24         self.received_dashStates = []
25
26     return self.current_state, self.next_state, self.reward, self.done, {}
```

In this sense, it should be noted that to induce the agent to learn how to orchestrate the switches queue sizes in a way to enhance the application Quality of Service (QoS), the reward score cannot be calculated immediately after the action taken in the current state, since the consequence of such an action would only be noticeable in the next state [11]. This phenomenon occurs because both TCP and the Adaptive Bitrate Streaming (ABR) algorithm have control mechanisms to alleviate the congestion on the network [46]. Hence, the reward calculation should be delayed until the next state observation. The implementation of this strategy can be observed in the conditional clause depicted in line 13 of Code 3.15.

In this context, the agent is rewarded for improving the video streaming QoS, characterized by FPS and the Local Buffer Occupancy (LBO). These metrics have an intrinsic correlation, such that as the LBO increases, there is a tendency for the FPS to increase as well. However, such a correlation is not always straightforward due to

the complex dynamics between network congestion and video streaming. Therefore, to calculate a reward (R_{t+1}) for a specific action (A_t), we can first evaluate whether the LBO from the next state (LBO_{t+1}) improved in comparison to the LBO observed when the action was executed (LBO_t) [11].

Then, a reward score is assigned considering the effects of this action on both LBO and FPS (FPS_{t+1}) next states. In this sense, the agent should receive a maximum reward whenever the action taken leads to the maximization of LBO_{t+1}, and is penalized in an inversely proportional manner if the video streaming present stalls (Code 3.16). This approach ensures that the agent's reward is conditioned by its ability to optimize both LBO and FPS, reducing the video streaming tradeoffs in dynamic network conditions [11]. Hence, when developing your agent, you should carefully design the reward function to reflect your application's intrinsic characteristics. This is important since the reward function affects the agent's learning capacity as much as the model hyperparameters.

Code 3.16: Calculate reward method from the Environment class

```
def calculate_reward(self, buffer_action_step, buffer_reward_step, fps_reward_step):
    # Check whether the LBO improved after the action taken
    if buffer_reward_step > buffer_action_step:
        if buffer_reward_step > 30:
            self.reward = 2
        # Check the next state's FPS and assign a reward score accordingly
        elif buffer_reward_step < 30:
            if fps_reward_step == 30:
                self.reward = 1
            elif fps_reward_step == 24:
                self.reward = .5
            else:
                self.reward = .1
    # Checking whether the LBO retarded after the action taken
    if buffer_reward_step < buffer_action_step:
        if buffer_reward_step > 30:
            self.reward = 2
        # Check the next state's FPS and assign a reward score accordingly
        elif buffer_reward_step < 30:
            if fps_reward_step == 30:
                self.reward = 1
            elif fps_reward_step == 24:
                self.reward = .5
            else:
                self.reward = -2
    # Append the calculated reward to the reward history
    self.reward_history.append(self.reward)
```

The *agent* in question is a Deep Q-Network (DQN) [36] designed with the Multi-Layer Perceptron (MLP) architecture, comprising an input layer with units for each INT feature, 2 hidden layers with 24 Rectified Linear Units (ReLU) each, and an output layer with 2 units - one for each action the agent could take (increase queue size to 64 or decrease it to 32). We trained an agent on the real setup throughout the video transmission to evaluate the feasibility of employing TimeGAN as a network traffic simulator for video applications. Afterward, we trained a second agent using only the synthetic data generated by TimeGAN. We followed the same methodology in both contexts. Still, in the latter one, instead of using the real setup, we fed the agent with the synthetic INT metadata corresponding to the action taken, thus simulating the real network behavior.

During the agent training, the actions to increase or decrease the switch's queue size are executed in accordance with the ε-greedy strategy, in which the agent starts ex-

ploring the environment by selecting random actions from the action space and exponentially decreases the probability to take such actions (ε) throughout the training steps to exploit the maximizing reward actions. Nonetheless, it should be noted that there must be a balance between exploration and exploitation to enable the agent to learn how to map actions to states continually. A well-adopted strategy is to start ε with 1.0 and exponentially decrease it to 0.01. Hence, we allow the agent to exploit its existing knowledge and explore new actions by intermittently choosing a random action from the action space [47]. Code 3.17 describes the exploration-exploitation strategy implementation.

Code 3.17: Epsilon-greedy policy method from the Agent class

```
def epsilon_greedy_policy(self, state):
    self.total_steps += 1
    # Is the probability of selecting a random action less than or equal to epsilon?
    if np.random.rand() <= self.epsilon:
        # If so, select a random action (exploration)
        return np.random.choice(self.num_actions)
    # Otherwise, select the action with the highest Q-value (exploitation)
    # Normalize the input state representation
    normalized_states = preprocessing.normalize(state)
    # Predict Q-values using the online network
    q = self.online_network.predict(normalized_states)
    # Return the action matching the highest Q-value
    return np.argmax(q, axis=1).squeeze()
```

In this context, we linearly decreased ε over a span of 250 time steps in order to favor exploration. Equation 2 formalizes the ε linear decay rate (D_{linear}) algorithm, which represents the ratio between the difference in starting (ε_s) and ending (ε_e) ε values, and the number of decay steps (T). Then, the probability of selecting random actions was exponentially reduced by a rate of 0.99 to a minimal value (0.01) in order to advance exploitation over exploration. Lines ranging from 9 to 15 of Code 3.18 describe the epsilon decay strategy implementation.

$$D_{linear} = (\varepsilon_s - \varepsilon_e) / T \tag{2}$$

Code 3.18: Memorize transition method from the Agent class

```
def memorize_transition(self, s, a, r, s_prime, not_done):
    # Check whether the episode has terminated
    if not_done == 0:
        # If the episode has terminated, update episode-related attributes
        self.episode_reward += r
        self.episode_length += 1
    # Otherwise, update the exploration rate based on training progress
    else:
        if self.train:
            if self.episodes < self.epsilon_decay_steps:
                # Decrease epsilon linearly
                self.epsilon -= self.epsilon_decay
            else:
                # Decrease epsilon exponentially
                self.epsilon *= self.epsilon_exponential_decay
    # Update the episode counter
    self.episodes += 1
    # Update the reward history
    self.rewards_history.append(self.episode_reward)
    # Update the episode length
    self.steps_per_episode.append(self.episode_length)
    # Reset episode-specific attributes
    self.episode_reward, self.episode_length = 0, 0
```

```
24      # Store the transition in the experience replay buffer
25      self.experience.append((s, a, r, s_prime, not_done))
```

The experiences resultant from agent-environment interactions are represented by a tuple of the current state, the action taken, the reward score, and the next state (`s`, `a`, `r`, and `s_prime` in Code 3.18). Once the agent parameters update condition is met, the DQN's online network is trained every time a new experience is stored in the experience replay memory buffer (lines ranging from 7 to 39 of Code 3.19). Conversely, the target network weights are updated at each τ step (line 49 from Code 3.19). For more information regarding the DQN training flow and the motivations supporting this methodology, please refer to the work by [36], since such content is beyond the scope of the current tutorial.

Code 3.19: Experience replay method from the Agent class

```
1  def experience_replay(self):
2      # If the experience replay does not meet the minimum requirement, no update occurs
3      if self.minimum_experience_memory > len(self.experience):
4          return
5      # Sample a minibatch of transitions from the experience replay buffer
6      minibatch = map(np.array, zip(*sample(self.experience, self.batch_size)))
7      states, actions, rewards, next_states, not_done = minibatch
8      # Normalize the states and next states
9      normalized_states = preprocessing.normalize(states)
10     normalized_next_states = preprocessing.normalize(next_states)
11     # Compute Q-values for next states using the online network
12     next_q_values = self.online_network.predict_on_batch(normalized_next_states)
13     # Select the best actions for next states
14     best_actions = tf.argmax(next_q_values, axis=1)
15     # Compute target Q-values using the target network
16     next_q_values_target = self.target_network.predict_on_batch(normalized_next_states)
17     # Gather target Q-values from the best actions considering the batch size
18     target_q_values = tf.gather_nd(next_q_values_target,
19                                    tf.stack((self.idx, tf.cast(best_actions, tf.int32)),
20                                    axis=1))
21     # Computes target Q-values based on the Bellman equation
22     targets = rewards + not_done * self.gamma * target_q_values
23     # Record the mean Q-value for monitoring
24     self.q_values.append(np.mean(targets))
25     # Predict the Q-values for the sampled batch of transitions
26     q_values = self.online_network.predict_on_batch(normalized_states)
27     # Update the Q-values for the sampled actions
28     q_values[self.idx, actions] = targets
29     # Train the online network using the updated Q-values
30     loss = self.online_network.train_on_batch(x=normalized_states, y=q_values)
31     self.losses.append(loss)
32     # Update the target network weights periodically
33     if self.total_steps % self.tau == 0:
34         self.update_target()
```

With that, we covered the first two objectives of this section: describing how to model an agent-environment interaction following the MDP strategy (Figure 3.2) and explaining how such an agent can learn to make assertive decisions in the context of queue management for video streaming applications. Now, we will describe how to employ the aforementioned methods. Thus, this use case is based on 3 Python scripts: `receiveMetricsGan.py`, which acts as our main script, by reading the INT metadata and controlling the agent training pipeline; `environmentGan.py`, which contains the functions to take actions and calculate their respective reward scores; and `agent.py`, which defines all components of the DQN workflow.

Hence, to apply the synthetic data generated by TimeGAN to train an RL-based agent, you should follow the following steps:

- Read INT metadata regarding the 32-bit and 64-bit queue sizes from their respective CSV files. In this step, you should instantiate a thread for each function in order to be able to read the data from both files simultaneously.
- Join the data obtained in the previous step into a global dictionary, this will enable us to change the data source in accordance with the agent actions, thus simulating the real setup's network behavior.
- Send data to the RL Environment. In this step, you should first verify whether the agent has taken actions and retrieve a data frame from the global dictionary based on the action taken or the switch's initial queue size. Then, you will need to slice this data frame and get the columns related to INT and DASH, and subsequently convert them to numpy arrays. This method will enable us to use these features as input for the DQN. As mentioned throughout this section, the INT metadata will be used by the agent for action-taking, while the DASH metrics will be employed in the agent's reward calculation.

The aforementioned steps should be repeated throughout the video streaming, which is the length of the synthetic data files. All code mentioned throughout this section is well documented and will be available in a GitHub repository[11].

3.5.2. Generation of Synthetic Network Traces

3.5.2.1. NetDiffusion

In this section, we will present a step-by-step guide[12] for synthetic traffic generation with a third-party tool namely NetDiffusion [25]. NetDiffusion is a state-of-the-art open-source tool for the community to generate traffic considering different applications. It leverages a controlled variant of a Stable Diffusion [43] model to generate synthetic network traffic that boasts high fidelity and adheres to protocol specifications. First of all, access your machine with GPU support, performing a port-forwarding on port 7860 and clone the `kohya_ss_fork` and `sd_webui_fork`. Then, activate the Python 3.10.13 virtual environment in `kohya_ss_fork` folder as follows:

Code 3.20: NetDiffusion: Clone the mandatory git forks

```
# SSH to Linux server via designated port (see following for example)
ssh -L 7860:LocalHost:7860 username@server_address
# Clone the repository
git clone git@github.com:noise-lab/NetDiffusion_Generator.git
# Navigate to the project directory
cd NetDiffusion_Generator
# Access the 'fine_tune' folder
cd fine_tune
# Remove the empty 'kohya_ss_fork @ 8a39d4d' and clone it as follows
git clone https://github.com/Chasexj/kohya_ss_fork.git
```

[11] https://github.com/thiagocaproni/tutorial_timegan/tree/master/code
[12] NetDiffusion tutorial: https://hackmd.io/@goes-ariel/rkVlaxTR6

```
# Clone 'sd-webui-fork @ 2533cf8' (also inside 'fine_tune' folder)
git clone https://github.com/Chasexj/stable-diffusion-webui-fork.git
cd <NetDiffusion main folder>/fine_tune/kohya_ss_fork/
source venv/bin/activate
```

So, We must ensure the variable `LD_LIBRARY_PATH` is correctly exported. Otherwise, a mandatory library, namely *bitsandbytes*, for CUDA custom functions - in particular 8-bit optimizers, matrix multiplication (LLM.int8()), and 8 & 4-bit quantization functions, will not work. First, we need to ensure what CUDA version is necessary for the virtual Python environment. To do that, just follow the steps in code 3.21:

Code 3.21: NetDiffusion: Checking Python's virtual environment CUDA version needed

```
#### Checking CUDA version ####
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/
    fine_tune/kohya_ss_fork$ python
Python 3.10.13 (main, Aug 25 2023, 13:20:03) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.version.cuda
'11.8'
>>> exit()
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/
    fine_tune/kohya_ss_fork$
#### Find libcudart.so library ####
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/
    fine_tune/kohya_ss_fork$ find / -name libcudart.so 2>/dev/null
... (omitted) ...
/home/thiago/anaconda3/envs/ydata/lib/libcudart.so
/home/thiago/anaconda3/pkgs/cuda-cudart-dev-12.1.105-0/lib/libcudart.so
/home/thiago/anaconda3/pkgs/cudatoolkit-11.3.1-h2bc3f7f_2/lib/libcudart.so
/home/thiago/anaconda3/pkgs/cudatoolkit-11.8.0-h6a678d5_0/lib/libcudart.so
... (omitted) ...
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/
    fine_tune/kohya_ss_fork$
#### Set the correct one (in this case, CUDA 11.8) ####
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/
    fine_tune/kohya_ss_fork$ export LD_LIBRARY_PATH=/home/thiago/anaconda3/pkgs/
    cudatoolkit-11.8.0-h6a678d5_0/lib
```

At lines 3-9, we verify the demanded version of CUDA for the Python's environment. Following, lines 13-22 depict the output of the existing CUDA versions and its respective `libcudart.so` libraries. In this case, we correctly set and export `LD_LIBRARY_PATH` (line 26).

Code 3.22: NetDiffusion: Checking whether bitsandbytes is running correctly

```
(venv) (base) thiago@ifsuldeminas-Z390-M-GAMING:~/git_ariel/NetDiffusion_Generator/
    fine_tune/kohya_ss_fork$ python -m bitsandbytes
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++ BUG REPORT INFORMATION ++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++ ANACONDA CUDA PATHS ++++++++++++++++++++
/home/thiago/anaconda3/lib/libicudata.so
/home/thiago/anaconda3/envs/ydata/lib/python3.9/site-packages/torch/lib/
    libtorch_cuda_linalg.so
... (ommited) ...

++++++++++++++++++ /usr/local CUDA PATHS +++++++++++++++++++
/usr/local/lib/python3.8/dist-packages/torch/lib/libc10_cuda.so
/usr/local/lib/python3.8/dist-packages/torch/lib/libtorch_cuda_linalg.so
/usr/local/lib/python3.8/dist-packages/torch/lib/libtorch_cuda.so

```

```
16 +++++++++++++++ WORKING DIRECTORY CUDA PATHS +++++++++++++++
17 /home/thiago/git_ariel/NetDiffusion_Generator/fine_tune/kohya_ss_fork/venv/lib/python3
       .10/site-packages/onnxruntime/capi/libonnxruntime_providers_cuda.so
18 /home/thiago/git_ariel/NetDiffusion_Generator/fine_tune/kohya_ss_fork/venv/lib/python3
       .10/site-packages/torch/lib/libtorch_cuda_linalg.so
19 /home/thiago/git_ariel/NetDiffusion_Generator/fine_tune/kohya_ss_fork/venv/lib/python3
       .10/site-packages/torch/lib/libtorch_cuda.so
20 ... (ommited) ...
21 ++++++++++++++++++ LD_LIBRARY CUDA PATHS ++++++++++++++++++
22  /home/thiago/anaconda3/pkgs/cudatoolkit-11.8.0-h6a678d5_0/lib CUDA PATHS
23 /home/thiago/anaconda3/pkgs/cudatoolkit-11.8.0-h6a678d5_0/lib/libcudart.so
24
25 ++++++++++++++++++++++++++ OTHER +++++++++++++++++++++++++++
26 COMPILED_WITH_CUDA = True
27 COMPUTE_CAPABILITIES_PER_GPU = ['7.5']
28 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
29 ++++++++++++++++++++ DEBUG INFO END ++++++++++++++++++++
30 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
31
32 Running a quick check that:
33     + library is importable
34     + CUDA function is callable
35
36 WARNING: Please be sure to sanitize sensible info from any such env vars!
37
38 SUCCESS!
39 Installation was successful!
```

In code block 3.22, we run the *bitsandbytes* module (line 2) and achieved the desired output (lines 66-74) where the import of the library is correct and the module is set. Otherwise, the output of this execution would indicate the need to export the `LD_LIBRARY_PATH` variable again.

To generate synthetic application data, we must convert application PCAPs/traces (e.g., Youtube, Skype) or the default provided Netflix traces into the nPrint [22] format, a standardized packet representation for Machine Learning model learning. To do that, we should store raw PCAPs used for fine-tuning into `NetDiffusion_Generator/data/fine_tune_pcaps` with the application/service labels as the filenames (e.g., `netflix_01.pcap`).

Code 3.23: NetDiffusion: Changing Caption

```
1 # For example, 'pixelated network data, type-0' refers to NetFlix pcap,
2 # Adjust the script based on fine-tuning task.
3 cd NetDiffusion_Generator/fine_tune && python3 caption_changing.py test_task/image/20
      _network
```

Code 3.24: NetDiffusion: Import Data

```
1 # Navigate to preprocessing dir
2 cd data_preprocessing/
3 # Run preprocessing conversions
4 python3 pcap_to_img.py
5 # Navigate to fine-tune dir and the khoya subdir for task creation
6 # (replace the number in 20_network with the average number of pcaps per traffic type
      used for fine-tuning)
7 cd ../fine_tune/kohya_ss_fork/model_training/
8 mkdir -p example_task/{image/20_network,log,model}
9 # Leverage Stable Diffusion WebUI for initial caption creation
10 cd ../../sd-webui-fork/stable-diffusion-webui/
11 # Lunch WebUI
12 bash webui.sh
```

The code section 3.24 summarizes to convert the nPrint data into an image representation (lines 2-5). Then, we create placeholder folders that will be used later. Also, we access a GUI (line 15) with the Python virtual environment and configure the fine-tuning parameters to train the model, as shown in the enumerated steps below. But first, we cannot forget the need to create a corresponding text file (e.g., "netflix_01.txt") with a customized traffic label for the preprocessed data (e.g., "netflix_01.png").

1. Open the WebUI via the ssh port on the preferred browser, example address: `http://localhost:7860/`
2. Under *Extras/Batch From Directory*, enter the absolute path for */NetDiffusion_Generator/data/preprocessed_fine_tune_imgs* and */NetDiffusion_Generator/fine_tune/kohya_ss_fork/model_training/test_task/image/20_network* as the *input/output* directories.
3. Under *Extras/batch_from_directory*, set the 'scale to' parameter to `width = 816` and *height = 768* for resource-friendly fine-tuning (adjust based on resource availability).
4. Enable the `caption` parameter under *extras/batch_from_directory* and click `generate`.
5. Terminate `webui.sh`

If you already run the setup.sh file inside `kohya_ss_fork` folder, do the following steps in code section 3.25 and check the installed requirements:

Code 3.25: NetDiffusion: Fine-tuning

```
# Navigate to fine-tuning directory
cd kohya_ss_fork
# Set up accelerate environment (gpu and fp16 recommended)
accelerate config
# Check the installed requirements with pip
pip list
# Fine-tune interface initialization
bash gui.sh
```

1. Open the fine-tuning interface via the ssh port on the preferred browser, example address: `http://localhost:7860/`
2. Under *LoRA/Training*, load the configuration file via the absolute path for */NetDiffusion_Generator/fine_tune/LoraLowVRAMSettings.json*
3. Under *LoRA/Training/Folders*, enter the absolute paths for */NetDiffusion_Generator/fine_tune/kohya_ss_fork/model_training/test_task/image*, */NetDiffusion_Generator/fine_tune/kohya_ss_fork/model_training/test_task/model*, and */NetDiffusion_Generator/fine_tune/kohya_ss_fork/model_training/test_task/log* for the Image/Output/Logging folders respectively, and adjust the model name if needed.
4. Under *LoRA/Training/Parameters/Basic*, adjust the Max Resolution (Figure 3.15) to match the resolution from data preprocessing, e.g., 816,768 (i.e., width, height).

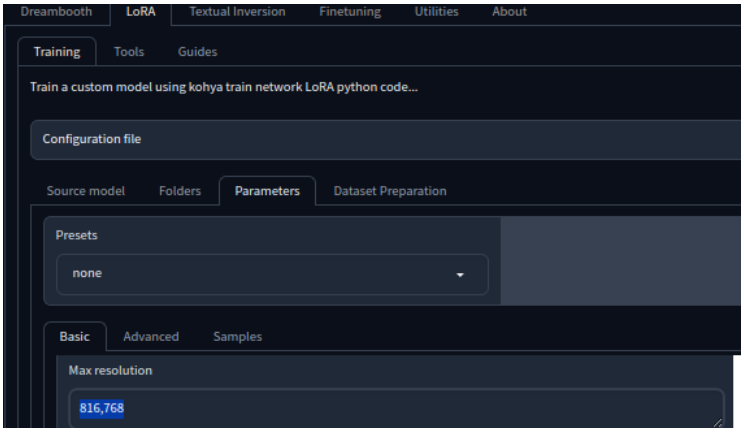


Figure 3.15: NetDiffusion GUI: fine-tuning max resolution

5. Click on Start Training to begin the fine-tuning. Adjust the fine-tuning parameters as needed due to different generation tasks that may have different parameter requirements to yield better synthetic data quality.
6. After the "model saved" message (check the terminal), close the server (Ctrl + C).

Figure 3.14 shows the GUI to configure the model paths and parameters. By default, the `LoraLowVRAMSettings.json` file, contains Windows-based file paths for *Image/Output/Logging* files. However, we can manually configure the correct absolute paths and the Max Resolution in this file and load it again.

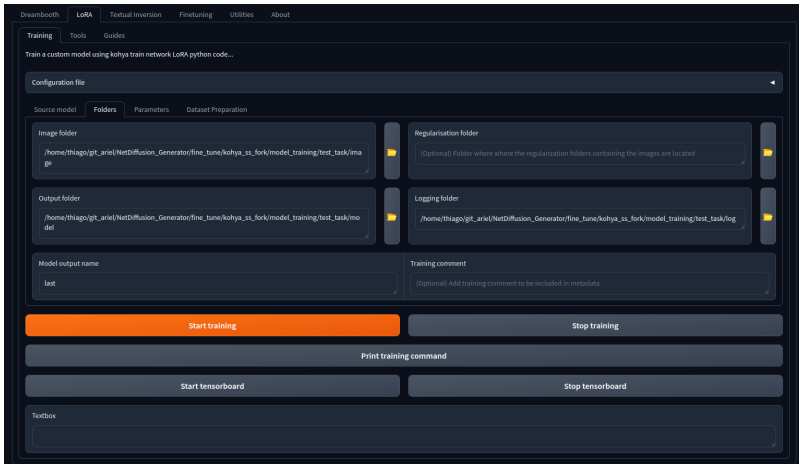


Figure 3.14: NetDiffusion GUI: fine-tuning folder paths.

In the previous steps we generated a file in the model folder provided (i.e., *fine-tune/kohya_ss_fork/model_training/test_task/model*). The default filename is `Addams.safetensors`. If not, replace "Addams" in the first step in the code block 3.26 by the correct name:

Code 3.26: NetDiffusion: Generation Step

```
# Copy the fine-tuned LoRA model (adjust path namings as needed) to Stable Diffusion WebUI
cp model_training/test_task/model/Addams.safetensors ../sd-webui-fork/stable-diffusion-webui/models/Lora/
# Navigate to the generation directory
cd ../sd-webui-fork/stable-diffusion-webui/
# Initialize Stable Diffusion WebUI
bash webui.sh
```

1. Open the WebUI via the ssh port on the preferred browser, example address: `http://localhost:7860/`
2. Install ControlNet extension for the WebUI and restart the WebUI: `https://github.com/Mikubill/sd-webui-controlnet` (see Figure 3.16). Go to *Extensions/* and place the URL in the appropriate field to install it.
3. To generate an image representation of a network trace, enter the corresponding caption prompt with the LoRA model extension under *txt2img* section. For example `pixelated network data, type-0 <lora:Addams:1>` for Netflix data.
4. Adjust the generation resolution to match the resolution from data preprocessing, e.g., `816,768`.
5. Adjust the seed to match the seed used in fine-tuning, default is `1234`.
6. Enable Hires.fix to scale to `1088, 1024`, upscaling the image (see Figure 3.18).
7. From training data, sample a real PCAP image (that belongs to the same category as the desired synthetic traffic) as input to the ControlNet interface, and set the Control Type (we recommend canny) - see Figure 3.17.
8. Click on `Generate` to complete the generation. Note that extensive adjustments on the generation and ControlNet parameters may be needed to yield the best generation result as the generation tasks and training data differ from each other.

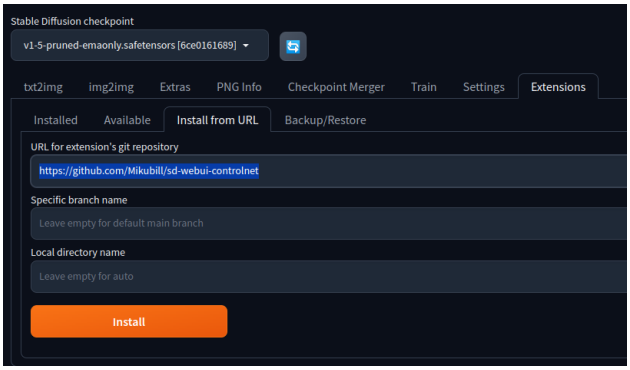


Figure 3.16: NetDiffusion: Installing ControlNet extension.

Remembering that the model name (with `.safetensors` extension) follows the same rule as in the previous steps. You can train different models with different names, but you must inform which one you are using in the prompt. For instance, consider we have a model with 3 applications (types 0 to 2), and we want to generate an

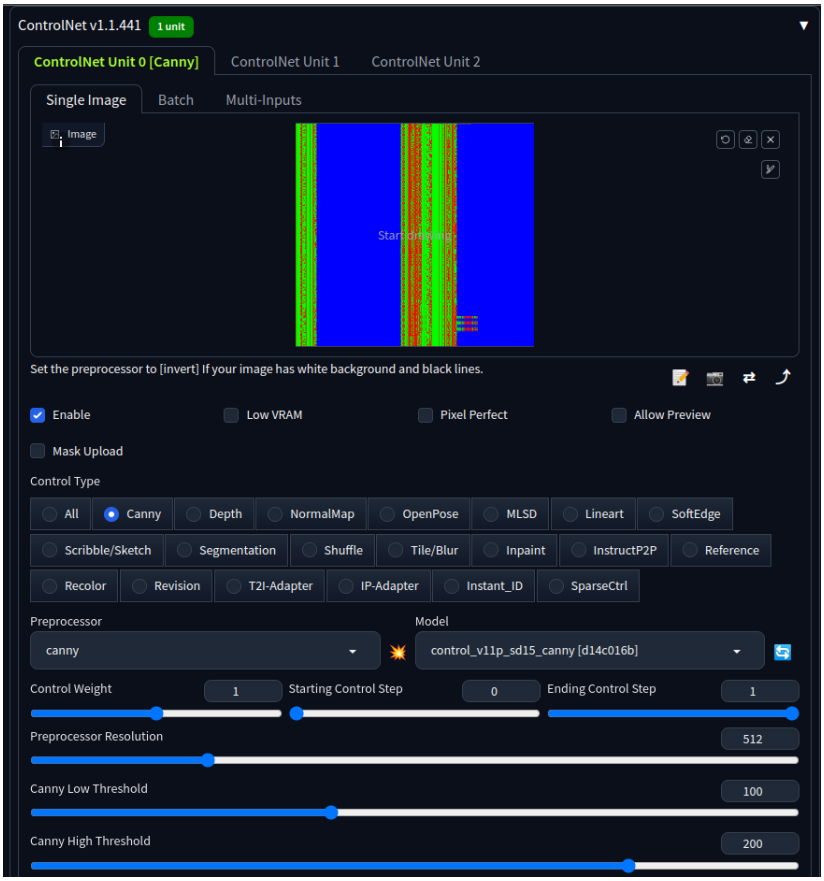


Figure 3.17: NetDiffusion: Configuring Canny filter parameters on ControlNet extension.

image representing a set of packets with the second class of applications. Then, we could set the prompt input as: `pixelated network data, type-1 <lora:my_model:1>`. To enable ControlNet, we must download it[13] and place it into *stable-diffusion-webui/extensions-webui-controlnet/models*

At this point, we already have a generated image. However, the generated image itself is not enough, as depicted in Figure 3.19, to reproduce a "replayable" PCAP (e.g., tcpreplay), since there is no guarantee the inter- and intra-packet dependencies are already being satisfied. In this case, the authors provided a set of post-heuristic scripts to convert the image back to nprint format and generate the final PCAP. First, we need to place the generated images into the correct folder - i.e., under */NetDiffusion_Generator/data/generated_imgs* – and navigate to the `/NetDiffusion_Generator/post-generation/` folder and run the post-generation scripts.

Figure 3.20 shows the execution of the post-heuristic processing. These steps complete the post-generation pipeline with the final nprints and PCAPs stored in */NetDiffusion_Generator/data/replayable_generated_nprints* and */NetDiffusion_Generator/data/replayable_generated_pcaps*, respectively.

To test the generated PCAPs, we can try to resend them through the local interface:

[13]Canny model (".pth" file)

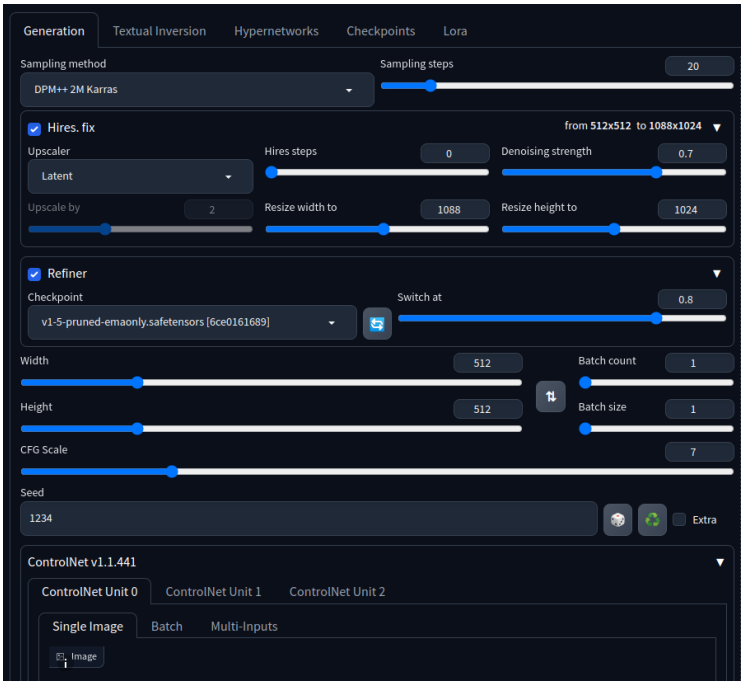


Figure 3.18: NetDiffusion: Upscaling the image to be generated.

Code 3.27: NetDiffusion: Post-Generation Heuristic Correction

```
# Install tcpreplay
sudo apt update
sudo apt install tcpreplay
# Grab the local interface's name
ip a
# Navidate to the generated PCAPs folder
cd NetDiffusion_Generator/data/replayable_generated_pcaps
# Run tcpreplay with a generated PCAP file
sudo tcpreplay --loop=0 --verbose -i eno1 00000-1234.pcap
```

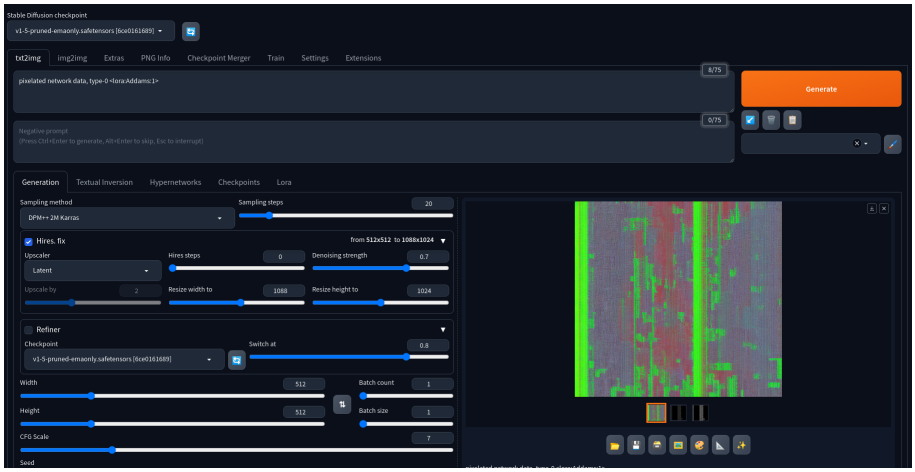


Figure 3.19: Generated image (without post-processing).

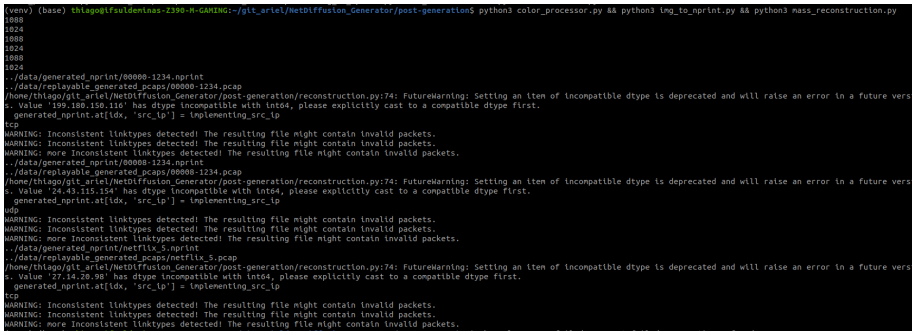


Figure 3.20: Running the post-processing heuristics.

3.5.2.2. NetShare

Similarly to NetDiffusion, NetShare [51] is a recent work exploring the feasibility of using GANs to automatically learn generative models to generate synthetic packet- and flow header traces for network-ing tasks (e.g., telemetry anomaly detection, provisioning). However, they do not use image representation or nPrint; rather, they use a tabular representation distributed among different GANs with temporal and confinement dependencies. The input packet/flow trace is split into a set of flows based on their 5-tuples, and each of these flows is further split into N chunks based on the time duration of the flow. NetShare divides each flow into 10 chunks. The packet/flow fields of these chunks are encoded using continuous and categorical encoding functions, and these ecoded chunks are fed into NetShare's DoppleGANger time series GANs, a specific variant of time series GANs that work well with time series data. Once trained, these models are used to generate synthetic (encoded) chunks, which are decoded and assembled into a full trace by sorting all the packets/flows by their timestamps

The work is well documented in terms of reproducibility. However, a direct way to convert the CSV output of key packet fields to PCAP and verify the usability of network packets with statistical similarities to real data to increase data availability in network experiments is not publicly available.

To install it, you may follow the step-guide installation tutorial on the official repository[14] or follow the next steps. First, we install the `libpcap-dev` library in the system - assuming we are on Ubuntu/Debian. Then, we create a Conda Python 3.9 virtual environment (line 6) and activate it (line 9). We then clone and install NetShare's dependencies (lines 12-13) and SDMetrics to help us plot and show the statistics after running the model (lines 16-17).

Code 3.28: **NetShare:** Setup Install

```
1 # Install libpcap depdency (Optional) - On Debian-based systems
2 sudo apt install libpcap-dev
3 # Assume Anaconda is installed
4 # Create virtual environment if not exists
5 conda create --name NetShare python=3.9
6 # Activate virtual env
7 conda activate NetShare
8 # Install NetShare package
```

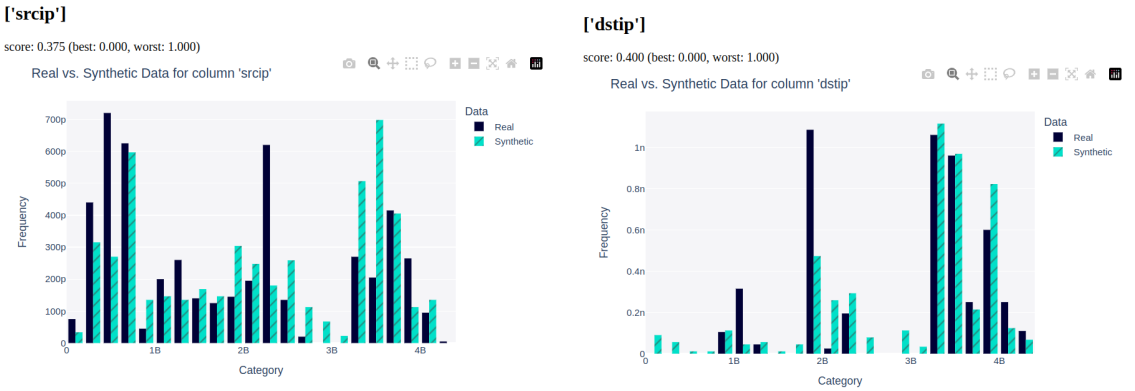
[14]NetShare GitHub Documentation: `https://github.com/netsharecmu/NetShare`

```
git clone https://github.com/netsharecmu/NetShare.git
pip3 install -e NetShare/
# Install SDMetrics package
git clone https://github.com/netsharecmu/SDMetrics_timeseries
pip3 install -e SDMetrics_timeseries/
```

```
Aggregated final dataset syndf
None (444, 12)
best_syn_df filename: ../../results/test-caida/post_processed_data/syn_df,dp_noise_multiplier-None,truncate-per_chunk,id-1.csv
Generated data is at ../../results/test-caida/post_processed_data
The filename with the largest ID is: syn_df,dp_noise_multiplier-None,truncate-per_chunk,id-1.csv
/home/thiago/git_ariel/SDMetrics_timeseries/sdmetrics/reports/utils.py:267: UserWarning:

Real or synthetic column session_length is a constant list. Not generating plots.

Dash is running on http://127.0.0.1:8050/

03/27/2024 15:46:56:INFO:Dash is running on http://127.0.0.1:8050/

 * Serving Flask app 'sdmetrics.reports.timeseries.quality_report'
 * Debug mode: off
03/27/2024 15:46:56:INFO:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:8050
03/27/2024 15:46:56:INFO:Press CTRL+C to quit
```

Figure 3.21: NetShare: Running the example - PCAP generation mode.

To run the example code with PCAP generation[15], go to `NetShare/examples/pcap` and run `driver.py`. As shown in Figure 3.21, NetShare opens a local connection with a GUI, where the results are visualized. There are several metrics (Figure 3.22) comparing how statistically similar the data is in various fields of the packet (e.g., source IP, destination IP).



(a) NetShare: Real vs Synthetic Source IP.

(b) NetShare: Real vs Synthetic Destiny IP.

Figure 3.22: NetShare: Some of the statistical results.

The results presented in Figure 3.22 are based on a sample CAIDA PCAP with only `76 KB` and do not represent the full potential of this work. The authors do provide a link to download the datasets[16] analyzed in the full paper for replicability. Although the code was available, the approach scales with compute resources, and the original models in the paper were trained on a 200 CPU + 200 GB Memory cluster, which are resources that neither we nor the authors have available at hand.

[15]NetShare also supports NetFlow data as input

[16]NetShare full-paper datasets: `https://drive.google.com/drive/folders/1FOl1VMr0tXhzKEOupxnJE9YQ2GwfX2FD?usp=sharing`

Figure 3.23: NetShare: CSV results.

Finally, the results shown in Fig. 3.23 demonstrate that the output is a CSV file, not PCAP, as expected. It is unclear whether any script from the authors has not been publicly available to convert the packet fields into CSV format. However, tools like PCAP Generator [9] can be used to convert the CSV, as long as each column is correctly formatted.

3.6. Conclusions and future perspectives

This tutorial book chapter has explored the multifaceted role of GANs in the context of Computer Networks, underscoring their potential to simulate complex network environments and generate synthetic data. Through analysis and practical applications, it has been shown how GANs can be employed to create realistic network data, aiding in training RL algorithms and enhancing network management operations.

We discussed the evolution of generative models, focusing on GANs and their application in producing high-fidelity synthetic network traffic, including PCAP files. We showcased GANs' ability to model and generate data that nearly mimics real network traffic, highlighting their significance in data augmentation, privacy preservation, and the development of robust network management solutions.

The tutorial contribution of this book chapter provided a hands-on approach to understanding the generation of synthetic time series data using GANs and its subsequent application in network telemetry within PDPs. The discussions and examples illustrated the practicality of GANs in creating synthetic datasets that can be leveraged for training ML models, particularly in environments where obtaining real, labeled datasets is challenging or privacy-sensitive.

Emerging trends in the field indicate a growing integration of generative AI models like GANs with network management and optimization tasks. Continuous advancement in this area promises innovative solutions to advance network systems' capabilities in handling dynamic, complex, and resource-intensive tasks.

In conclusion, this chapter's exploration of GANs is a step toward understanding impact opportunities in the field of Computer Networks. Multiple challenges are open for research and development of synthetic data generation, network simulation, and the holistic integration of AI in network management, aiming to achieve more intelligent, autonomous, and efficient network systems.

Acknowledgments

This work has been supported by the following Brazilian research agencies: Federal Institute of Education, Science, and Technology of South of Minas Gerais - IFSULDEMINAS,

FAPESP, and CAPES. This study was partially funded by CAPES, Brazil - Finance Code 001. This work was partially supported by the Innovation Center, Ericsson S.A., and by the Sao Paulo Research Foundation (FAPESP), grant 2021/00199-8, CPE SMARTNESS.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[2] Leandro Almeida, Jefferson Silva, Ricardo Lins, Paulo Maciel Jr., Rafael Pasquini, and Fábio Verdi. Wave - um gerador de cargas múltiplas para experimentação em redes de computadores. In *Anais Estendidos do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 9–16, Porto Alegre, RS, Brasil, 2023. SBC.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[4] Serhat Arslan and Nick McKeown. Switches know the exact amount of congestion, 2019.

[5] Raef Bassily, Albert Cheu, Shay Moran, Aleksandar Nikolov, Jonathan Ullman, and Steven Wu. Private query release assisted by public data. In *International Conference on Machine Learning*, pages 695–703. PMLR, 2020.

[6] Jay Beale, Angela Orebaugh, and Gilbert Ramirez. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[7] Eoin Brophy, Zhengwei Wang, Qi She, and Tomás Ward. Generative adversarial networks in time series: A systematic literature review. *ACM Comput. Surv.*, 55(10), feb 2023.

[8] Tobias Bühler, Roland Schmid, Sandro Lutz, and Laurent Vanbever. Generating representative, live network traffic out of millions of code repositories, 2022.

[9] Levente Csikor. Pcap generator. `https://github.com/cslev/pcap_generator`, 2017. Accessed: 2024/03/10.

[10] Enyan Dai and Suhang Wang. Say no to the discrimination: Learning fair graph neural networks with limited sensitive attribute information, 2021.

[11] Leandro C de Almeida et al. Desired - dynamic, enhanced, and smart ired: A p4-aqm with deep reinforcement learning and in-band network telemetry. *Computer Networks*, 244(110326):1–19, 2024.

[12] Baik Dowoo, Yujin Jung, and Changhee Choi. Pcapgan: Packet capture file generator by style-based generative adversarial networks. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1149–1154. IEEE, 2019.

[13] Farnoush Falahatraftar, Samuel Pierre, and Steven Chamberland. A conditional generative adversarial network based approach for network slicing in heterogeneous vehicular networks. *Telecom*, 2(1):141–154, 2021.

[14] Luis Fernando Uria Garcia, Rodolfo S. Villaça, Moisés R. N. Ribeiro, Regis Francisco Teles Martins, Fábio Luciano Verdi, and Cesar Marcondes. Introdução à linguagem p4 - teoria e prática. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC) - Minicursos*, 2018.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets, 2014.

[16] Carroll Gray-Preston. Ai network applications, Sep 2023.

[17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[18] Rohit Kumar Gupta, Shashwat Mahajan, and Rajiv Misra. Resource orchestration in network slicing using gan-based distributional deep q-network for industrial applications. *The Journal of Supercomputing*, 79(5):5109–5138, 2023.

[19] Ariel Góes de Castro, Fábio Rossi, Arthur Lorenzon, Marcelo Caggiani Luizelli, Francisco Vogt, Rumenigue Hohemberger, and Rodrigo Mansilha. Orchestrating in-band data plane telemetry with machine learning. *IEEE Communications Letters*, 23:20, 10 2019.

[20] Luchao Han, Yiqiang Sheng, and Xuewen Zeng. A packet-length-adjustable attention model based on bytes embedding using flow-wgan for smart cybersecurity. *IEEE Access*, 7:82913–82926, 2019.

[21] Frederik Hauser, Marco Häberle, Daniel Merling, Steffen Lindner, Vladimir Gurevich, Florian Zeiger, Reinhard Frank, and Michael Menth. A survey on data plane programming with p4: Fundamentals, advances, and applied research. *Journal of Network and Computer Applications*, 212:103561, 2023.

[22] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. New directions in automated traffic analysis. CCS '21, page 3366–3383, New York, NY, USA, 2021. Association for Computing Machinery.

[23] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[24] Yuxiu Hua, Rongpeng Li, Zhifeng Zhao, Honggang Zhang, and Xianfu Chen. Gan-based deep distributional reinforcement learning for resource management in network slicing, 2019.

[25] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Arjun Nitin Bhagoji, Paul Schmitt, Francesco Bronzino, and Nick Feamster. Netdiffusion: Network data augmentation through protocol-constrained traffic generation. *arXiv preprint arXiv:2310.08543*, 2023.

[26] Alexey Kurakin, Shuang Song, Steve Chien, Roxana Geambasu, Andreas Terzis, and Abhradeep Thakurta. Toward training at imagenet scale with differential privacy. *arXiv preprint arXiv:2201.12328*, 2022.

[27] Gwo-Chuan Lee, Jyun-Hong Li, and Zi-Yang Li. A wasserstein generative adversarial network–gradient penalty-based model with imbalanced data enhancement for network intrusion detection. *Applied Sciences*, 13(14):8132, 2023.

[28] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions, 2020.

[29] Qiang Liu, Nakjung Choi, and Tao Han. Atlas: Automate online service configuration in network slicing. In *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '22, page 140–155, New York, NY, USA, 2022. Association for Computing Machinery.

[30] Pranav Mani, E. S. Gopi, Hrishikesh Shekhar, and Sharan Chandra. Generative adversarial network and reinforcement learning to estimate channel coefficients. In E. S. Gopi, editor, *Machine Learning, Deep Learning and Computational Intelligence for Wireless Communication*, pages 49–58, Singapore, 2021. Springer Singapore.

[31] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[32] Amar Meddahi, Hassen Drira, and Ahmed Meddahi. Sip-gan: Generative adversarial networks for sip traffic generation. In *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2021.

[33] Jiayi Meng, Jingqi Huang, Y Charlie Hu, Yaron Koral, Xiaojun Lin, Muhammad Shahbaz, and Abhigyan Sharma. Characterizing and modeling control-plane traffic for mobile core network. *arXiv preprint arXiv:2212.13248*, 2022.

[34] Jesse G Meyer, Ryan J Urbanowicz, Patrick CN Martin, Karen O'Connor, Ruowang Li, Pei-Chen Peng, Tiffani J Bright, Nicholas Tatonetti, Kyoung Jae Won, Graciela Gonzalez-Hernandez, et al. Chatgpt and large language models in academia: opportunities and challenges. *BioData Mining*, 16(1):20, 2023.

[35] Qinghai Miao, Yisheng Lv, Min Huang, Xiao Wang, and Fei-Yue Wang. Parallel learning: Overview and perspective for computational learning across syn2real and sim2real. *IEEE/-CAA Journal of Automatica Sinica*, 10(3):603–631, 2023.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning, 2015.

[37] Muhammad Haris Naveed, Umair Sajid Hashmi, Nayab Tajved, Neha Sultan, and Ali Imran. Assessing deep generative models on time series network data. *IEEE Access*, 10:64601–64617, 2022.

[38] Hojjat Navidan, Parisa Fard Moshiri, Mohammad Nabati, Reza Shahbazian, Seyed Ali Ghorashi, Vahid Shah-Mansouri, and David Windridge. Generative adversarial networks (gans) in networking: A comprehensive survey & evaluation, 2021.

[39] P4. In-band network telemetry (int) dataplane specification. Technical report, P4 Consortium, 2021.

[40] Cheng Qian, Wei Yu, Chao Lu, David Griffith, and Nada Golmie. Toward generative adversarial networks for the industrial internet of things. *IEEE Internet of Things Journal*, 9(19):19147–19159, 2022.

[41] Haneya Naeem Qureshi, Usama Masood, Marvin Manalastas, Syed Muhammad Asad Zaidi, Hasan Farooq, Julien Forgeat, Maxime Bouton, Shruti Bothe, Per Karlsson, Ali Rizwan, et al. Towards addressing training data scarcity challenge in emerging radio access networks: A survey and framework. *IEEE Communications Surveys & Tutorials*, 2023.

[42] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. *Computers & Security*, 82:156–172, 2019.

[43] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[44] Mustafizur R Shahid, Gregory Blanc, Houda Jmila, Zonghua Zhang, and Hervé Debar. Generative deep learning for internet of things network traffic generation. In *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 70–79. IEEE, 2020.

[45] Jacob Soper, Yue Xu, Kien Nguyen, Ernest Foo, and Zahra Jadidi. A two-pass approach for minimising error in synthetically generated network traffic data sets. In *Proceedings of the 2023 Australasian Computer Science Week*, pages 18–27. 2023.

[46] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. From theory to practice: improving bitrate adaptation in the dash reference player. In *ACM Transactions on Multimedia Computing, Communications, and ApplicationsVolume 15Issue 2s*, pages 123–137, 06 2018.

[47] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[48] Mohammad Mustafa Taye. Understanding of machine learning with deep learning: Architectures, workflow, applications and future directions. *Computers*, 12(5), 2023.

[49] Zehui Xiong, Yang Zhang, Dusit Niyato, Ruilong Deng, Ping Wang, and Li-Chun Wang. Deep reinforcement learning for mobile 5g and beyond: Fundamentals, applications, and challenges, 2019.

[50] Shengzhe Xu, Manish Marwah, Martin Arlitt, and Naren Ramakrishnan. Stan: Synthetic network traffic generation with generative neural models. *arXiv preprint arXiv:2009.12740*, 2020.

[51] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. Practical gan-based synthetic ip header trace generation using netshare. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 458–472, New York, NY, USA, 2022. Association for Computing Machinery.

[52] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks, 2019.

[53] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[54] Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, et al. Differentially private fine-tuning of language models. *arXiv preprint arXiv:2110.06500*, 2021.

[55] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023.

[56] Cong Zou, Fang Yang, Jian Song, and Zhu Han. Generative adversarial network for wireless communication: Principle, application, and trends. *IEEE Communications Magazine*, pages 1–7, 2023.

Chapter 4

Desaprendizado de Maquinas e a LGPD: da privacidade ao direito ao esquecimento

Daniel O. C. Cota, Daniel Carlos S. de Jesus, Antonio A. de A. Rocha

Abstract

By voluntarily providing huge amounts of personal data to online services such as Facebook, Google and Amazon in exchange for useful services, users are often unaware of the great risk they are taking on. The security aspect associated with access to users' personal information has been a cause for concern, as it can pose a threat to their privacy. Against this backdrop, regulations have given users the option of removing their data from a system, such as the "right to be forgotten". But what does deleting data mean? Normally, data is not isolated in a database, it is used, for example, to train machine learning models. Deleting a user's data may be enough to prevent it from influencing the training of future models, but it doesn't eliminate the influence of the data on existing models. Faced with this challenge, the machine unlearning paradigm has emerged, consisting of algorithms/frameworks that allow selective forgetting of data, comparing the distributions of the unlearned model with the retrained one.
Keywords*: Machine Learning, Machine unlearning, privacy.*

Resumo

Ao fornecer voluntariamente enormes quantidades de dados pessoais a serviços online, como o Facebook, o Google e a Amazon, em troca de serviços úteis, os usuários muitas vezes não estão cientes do grande risco que assumem. O aspecto da segurança associado ao acesso as informações pessoais dos usuários tem sido alvo de preocupação, visto que pode representar uma ameaça a sua privacidade. Diante de tal cenário, regulamentações tem proporcionado aos usuários a possibilidade de optar pela remoção dos seus dados de um sistema, como o intitulado "o direito a serem esquecidos". Mas o que é que significa apagar dados? Normalmente, os dados não ficam isolados numa base de dados, eles são utilizados, por exemplo, para treinamento de modelos de aprendizado de máquinas. A eliminação dos dados de um usuário podem ser suficientes para impedir que

estes influenciem a formação de modelos futuros, mas não elimina a influência dos dados nos modelos existentes. Diante deste desafio, surge o paradigma do desaprendizado de máquinas, do inglês machine unlearning, que consiste em algoritmos/frameworks que permitem o esquecimento seletivo de dados, comparando a distribuições do modelo desaprendido com o retreinado.
***Palavras-chaves**: Machine Learning, Machine unlearning, privacidade.*

4.1. O que é desaprendizado?

Ao longo dos anos, a sociedade tem sido moldada pela aprendizagem, que pode ser entendida como um processo no qual o ser humano modifica suas habilidades e conhecimentos a partir de experiência direta, estudo, observação, raciocínio ou instrução. Entretanto, por inúmeros fatores é possível afirmar que nem toda informação aprendida tem um resultado positivo, sendo necessário o seu esquecimento.

O ato de desaprender que pode definido como o processo de perda ou esquecimento daquilo que aprendera, que sabia [Ferreira 1999], tem sido alvo de estudos, dentre as principais áreas é possível citar nas organizações. O entitulado desaprendizado organizacional pode ser entendido da seguinte maneira, da mesma forma que as organizações desenvolvem processos para que o aprendizado possa ocorrer e ser acumulado, também precisam buscar meios pelos quais os aprendizados passados possam ser revistos. O trabalho de [Holan and Phillips 2004] define esquecimento organizacional como "a perda, voluntária ou não, do conhecimento organizacional". Os autores ainda afirmam que desaprendizagem consiste em um tipo de esquecimento, que ocorre quando o conhecimento organizacional é intencionalmente removido [Buchele et al. 2016].

Assim como nas organizações, o processo de esquecimento intencional das informações dos indivíduos na atual "era dos dados", com o desenvolvimento da Lei Geral de Proteção dos Dados (LGPD), tem gerado um aumento das atenções para lidar com este problema, especialmente por envolver aspecto da privacidade do indivíduo. A estratégia que trata desta problemática e que será alvo deste estudo é conhecida como desaprendizagem de máquina.

4.1.1. O que é desaprendizado de máquina (DM)?

Nos últimos anos, sociedade tem presenciado o notório desenvolvimento das redes móveis, especialmente com o advento das tecnologias 5G e 6G. A evolução tecnológica e o acesso cada vez mais facilitado a dispositivos eletrônicos e à conectividade têm impulsionado o uso crescente destes aparelhos. Hoje, é comum deparar-se com pessoas de diversas faixas etárias utilizando dispositivos computadorizados e navegando na internet para trabalhar, estudar, se comunicar, realizar compras, buscar informações e se entreter. Essas atividades produzem um elevado número de dados pessoais que vão deixando rastros da Internet que podem refletir comportamentos, preferências e interações dos usuários.

A coleta de volume massivo de dados pessoais como: nome, apelido, localização, e-mail, histórico de navegação, informações sobre renda e endereço de IP tem sido utilizada para o avanço da inteligência artificial, especialmente dos modelos de aprendizado de máquinas. Essas informações disponibilizadas pelos conhecidos cookies, tecnologia que fornece "a digital do usuário", possibilitam que os algoritmos utilizados para treinar

os modelos sejam capazes de encontrar padrões de comportamento em função de características em comum dos usuários avaliados. Vale ressaltar que esses "insights" são extremamente valiosos para as grandes corporações, visto que fornecem informações importantes sobre os seus clientes.

Entretanto, o aspecto da segurança associado ao acesso as informações pessoais dos usuários tem sido alvo de preocupação, visto que pode representar uma ameaça a sua privacidade. Diante de tal cenário, os usuários podem optar pela remoção dos seus dados de um sistema, conforme prevê o artigo 17º do Regulamento Geral de Proteção de Dados (RGPD) que obriga as organizações a fornecer aos usuários "o direito a serem esquecidos", ou seja, o direito a que todos ou parte dos seus dados sejam eliminados de um sistema mediante solicitação.

Embora a remoção de dados das bases de dados back-end atenda a nova regulamentação, o mesmo não é suficiente no contexto da IA, uma vez que os modelos de aprendizado de máquinas são criados a partir da compressão dos dados de treinamento, sendo que alguns são demasiadamente adaptados ao seu treino. Além disso, há modelos mais complexos como de aprendizado profundo na qual é difícil identificar a ligação entre os dados e os parâmetros do modelo, mas é possível observar o notório impacto que tal remoção terá sobre a sua performance. Há ainda a opção de retreinar o modelo retirando os dados solicitados, entretanto o processo de retreinamento caracteriza-se por ser computacionalmente dispendioso, o que representa um entrave importante para a sua utilização [Nguyen et al. 2022].

Logo, é evidente o desafio de estabelecer uma estratégia que permite ao modelo de aprendizado de máquina “esquecer” informações previamente aprendidas através dos dados de treinamento. Este processo de remover seletivamente a influência de dados específicos nos modelos treinados é chamada de desaprendizado de máquinas (DM), conhecido pelo seu termo em inglês "machine unlearning".

4.2. Workflow do processo de DM

A Figura 4.1 apresenta o fluxo de trabalho típico de desaprendizado de máquinas, que basicamente pode ser representado por um modelo de aprendizado de máquinas acrescido da presença de um pedido de remoção de dados.

O workflow de desaprendizagem apresenta o fluxo de trabalho típico de um modelo de aprendizado de máquinas na presença de um pedido de remoção de dados. Em geral, um modelo é criado, a partir dos dados de treinamento e em seguida é utilizado para inferência. Após um pedido de remoção, os dados a serem esquecidos são retirados do modelo. O modelo desaprendido é então verificado em relação a critérios de privacidade e, se estes critérios não forem cumpridos, o modelo é treinado de novo, ou seja, se o modelo ainda deixar escapar alguma informação sobre os dados esquecidos. Existem dois componentes principais neste processo: o componente de aprendizagem (à esquerda) e o componente de desaprendizagem (à direita).

A componente de aprendizagem envolve os dados atuais, um algoritmo de aprendizagem e o modelo atual. No início, o modelo inicial é treinado a partir de todo o conjunto de dados utilizando o algoritmo de aprendizagem. A componente de desapren-

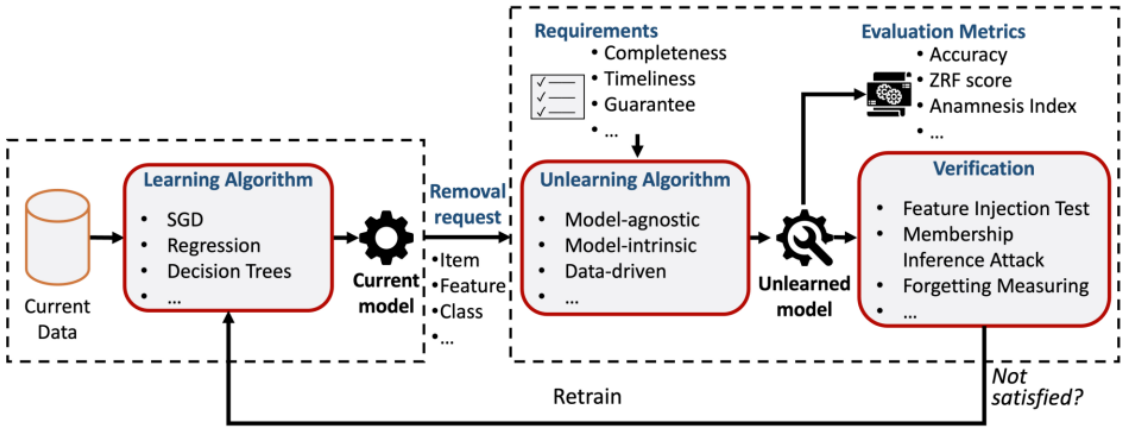


Figure 4.1. Workflow do processo de DM [Nguyen et al. 2022]

dizagem envolve um algoritmo de desaprendizagem, o modelo desaprendido, requisitos de otimização, métricas de avaliação e um mecanismo de verificação. Após um pedido de remoção de dados, o modelo atual será processado por um algoritmo de desaprendizagem para esquecer a informação correspondente a esses dados dentro do modelo. O algoritmo de desaprendizagem pode ter em conta vários requisitos, como a exaustividade, a atualidade e as garantias de privacidade. O resultado é um modelo não aprendido, que será avaliado em função de diferentes métricas de desempenho (por exemplo, completude, pontuação ZRF, índice de anamnese). No entanto, para fornecer um certificado de privacidade para o modelo não aprendido, é necessária uma verificação (ou auditoria) para provar que o modelo esqueceu efetivamente os dados solicitados e que não há fugas de informação. Esta auditoria pode incluir um teste de injeção de características, um ataque de inferência de associação ou medidas de esquecimento. Se o modelo não aprendido passar na verificação, torna-se o novo modelo para tarefas em questão, que podem ser de inferência, previsão, classificação ou recomendação. Se o modelo não passar na verificação, os dados restantes, ou seja, os dados originais excluindo os dados a serem esquecidos, precisam ser utilizados para treinar novamente o modelo. De qualquer forma, o componente de desaprendizagem será chamado repetidamente por meio de um novo pedido de remoção [Nguyen et al. 2022].

4.2.1. Definição formal de AM

O conceito de aprendizado de máquinas já está bastante consolidado na literatura. Dentre os trabalhos que abordam o tema, é possível citar o artigo de [Langley and Carbonell 1984] que o define como um domínio que busca desenvolver métodos e técnicas para automatizar a aquisição de novas informações, novas competências e novas formas de organizar as informações existentes.

A Figura 4.2 representa o diagrama do processo de aprendizado supervisionado, no qual é possível verificar que dentre do conjunto de hipóteses H, o objetivo do aprendizado supervisionadp é encontrar a melhor h, denominada hipótese final g, que de alguma forma se aproxime da função alvo f. Para tal, é necessário definir o algoritmo de aprendizagem A, que inclui a função objetivo (a função a otimizar para procurar g) e os

métodos de otimização [Chao 2011].

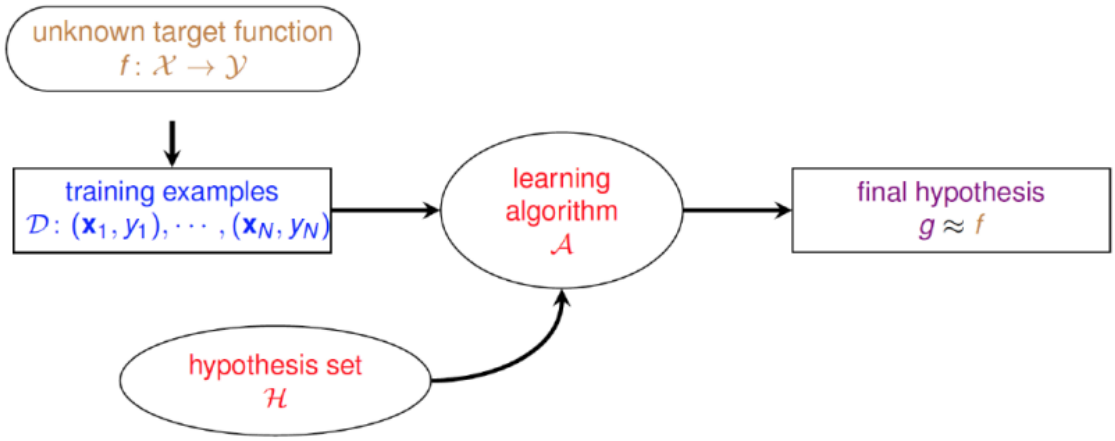


Figure 4.2. Diagrama do processo de aprendizado de máquinas

4.2.2. Definição formal de DM

Como a desaprendizagem de máquina parte do pressuposto que o modelo já "aprendeu", para suportar os pedidos de esquecimento o sistema computacional precisa de um mecanismo de desaprendizagem, denotada pela função U, que recebe como entrada um conjunto de dados de treino $D \in Z^*$, em que Z^* representa o conjunto de todos os datasets de treino possíveis. Além de um conjunto de esquecimento $D_f \subset D$ e um modelo de aprendizado A(D). O resultado será um modelo "desaprendido" $U(D, D_f, A(D)) \in H$, que tem como hipótese final ser um modelo que se aproxima do modelo retreinado $A(D|D_f)$. As principais notações que envolvem o processo de desaprendizado de máquinas podem ser visto na Figura 4.3.

Portanto, é possível dizer que na prática o problema central do desaprendizado de máquina envolve a comparação entre duas distribuições de modelos de aprendizado de máquina, o desaprendido e o retreinado [Bourtoule et al. 2021, Brophy and Lowd 2021, Thudi et al. 2022].

4.3. Por que desaprender?

Em 2010, foi desenvolvido um primeiro projeto legislativo na França que previa a criação de um "direito a ser esquecido" online. Posteriormente, não foram disponibilizadas muitas informações concretas sobre a proposta de lei, que pretendia obrigar as empresas do setor tecnológico a eliminarem as mensagens de correio eletrônico e de texto após um período de tempo acordado ou a pedido da pessoa em causa.Em novembro de 2010, a Comissão Europeia retomou a ideia de introduzir um direito a ser esquecido no contexto proteção de dados; o resultado da vaga proposta é ainda incerto [Weber 2011].

As discussões a respeito do tema retornaram com maior enfâse no ano de 2018, em virtude de um dos maiores escândalos no mundo da tecnologia, protagonizado pela Cambridge Analytica, uma empresa de marketing que teve acesso a dados de 87 milhões de usuários do Facebook indevidamente. Desde então, aspectos envolvendo segurança e privacidade dos dados, como as disposições e as sanções em matéria de proteção de

Symbols	Definition
$\mathcal{Z}$	example space
D	the training dataset
D_f	forget set
$D_r = D \setminus D_f$	retain set
$A(.)$	a learning algorithm
$U(.)$	an unlearning algorithm
$\mathcal{H}$	hypothesis space of models
$w = A(D)$	Parameters of the model trained on D by A
$w_r = A(D_r)$	Parameters of the model trained on D_r by A
$w_u = U(.)$	Parameters of the model unlearned by $U(.)$

Figure 4.3. Notações importantes do processo de desaprendizado de máquinas

dados, tem sido considerado internacionalmente como o correto caminho a seguir.

Nos últimos anos, foram promulgados muitos regulamentos visando a preservação da privacidade, que envolvem o "direito a ser esquecido" [Dang 2021]. Dentre as principais regulamentações, destacam-se o Regulamento Geral sobre a Proteção de Dados (RGPD) da União Europeia [Magdziarczyk 2019], na qual é aplicada sempre que os dados pessoais são processados, o que inclui o seu recolhimento, transformação, consulta ou eliminação, dentro ou fora da União Europeia (UE), bastando que os dados digam a respeito a um residente da UE [Veale et al. 2018]. Dentre outros regulamentos bastante conhecidos é possível citar a Lei da Privacidade do Consumidor da Califórnia (CCPA) [Pardau 2018], que estabelece que os usuários devem ter o direito de apagar os seus dados e informações relacionadas para proteger a sua privacidade. Em parte, esta legislação surgiu na sequência de fugas de informação sobre a privacidade. Por exemplo, casos de fugas de dados dos usuários dos sistemas de computação em nuvem, devido a múltiplas cópias dos dados mantidas por diferentes partes, políticas de cópia de segurança e estratégias de replicação.

Já no Brasil, a regulamentação se deu com a edição da Lei 13.709/2018, a denominada Lei Geral de Proteção de Dados (LGPD). A LGPD coloca o indivíduo como protagonista das relações jurídicas que envolvam o tratamento de dados, não apenas por regular a proteção de dados pessoais, mas, principalmente, elege como fundamento em seu art. 2º, II, a "autodeterminação informativa", que confere a pessoa o direito de escolher quais dados serão usados, bem como os limites e o prazo de sua utilização [Capanema 2020].

Embora seja um tema recente, a preocupação em saber se os sistemas de aprendizagem de máquina estão suficientemente regulamentados tem ganhado destaque no direito e nas políticas envolvendo o setor tecnológico. Dentre os elementos agravadores é possível citar o seu aparente potencial para reproduzir a discriminação social e transformar dados pessoais desprentensiosos em informações sensíveis [Veale et al. 2018]. Diante deste cenário, o processo de desaprendizagem torna-se estritamente necessário.

4.3.1. Motivação: Por que utilizar DM?

Além da questões envolvendo privacidade, é possível citar inúmeras razões pelas quais um usuário pode querer eliminar os seus dados de um sistema, tais motivos podem ser divididos entre grupos a seguir:

- Segurança: Apesar da importância de modelos de aprendizado profundo para o desenvolvimento do uso de IA em problemas do cotidiano, pesquisas recentes apontam que esses modelos são vulneráveis a ataques cibernéticos [Ren et al. 2020]. Dentre as principais estratégias maliciosas utilizadas é possível citar o envenenamento de dados. Para manipular o comportamento do modelo de machine learning (ML), o envenenamento de dados consiste em adicionar dados maliciosos aos conjuntos de dados de treinamento. Estes dados caracterizam-se por ser extremamente semelhantes aos dados originais, a ponto de um ser humano não conseguir distinguir entre os dados reais e os falsos. O principal objetivo deste ataque é manipular o comportamento do modelo treinado e fornecer resultados falsos, o que pode resultar em problemas graves. Por exemplo, na área da saúde, uma previsão errada pode levar a um diagnóstico errado, e consequentemente a um tratamento inadequado, que pode ocasionar a morte do paciente. Logo, a detecção e remoção de dados contraditórios é essencial para garantir a segurança do modelo e, uma vez detectado um ataque, o modelo tem de ser capaz de eliminar os dados contraditórios através de um mecanismo de desaprendizado de máquinas [Cao and Yang 2015, Marchant et al. 2022].

- Usabilidade: Os sistemas de recomendação apontam para as diferentes preferências em aplicações ou serviços onlines para cada usuário. Entretanto, uma aplicação produzirá recomendações inconvenientes se não puder eliminar completamente os dados incorretos relacionados ao usuário, frutos por exemplo, de ruído ou dados maliciosos. Suponha que uma pessoa tenha acidentalmente procurado por um produto ilegal no seu computador e descobriu que continua a receber a recomendação desse produto, mesmo depois de ter apagado o histórico do seu navegador [Cao and Yang 2015]. Esta usabilidade indesejada por não esquecer os dados não só produzirá previsões erradas, como também resultará em menos usuários do sistema. Neste cenário, a desaprendizagem de máquina seria uma alternativa viável para fazer o sistema esquecer esta informação.

- Justiça (Fairness) / IA Ética: No contexto atual, é essencial desenvolver sistemas de forma responsável, que levam em consideração não apenas a eficiência e o desempenho, mas também as implicações éticas e sociais dessas ferramentas, uma vez que elas impactam diretamente a vida das pessoas. Apesar dos avanços recentes, os modelos de aprendizado de máquinas continuam a ser sensíveis a preconceitos, o que significa que os seus resultados podem discriminar injustamente um grupo de pessoas [Mehrabi et al. 2021]. Logo, diante de modelos que foram treinados com algum tipo de viés é necessário desaprender estes dados, incluindo os atributos dos dados afetados.

- Otimização do desempenho: Há ainda o aspecto envolvendo a performance do modelo de aprendizado de máquinas, que dentre outros fatores podem ter desempenho

comprometido em virtude da influência de amostras de dados de baixa qualidade. As amostras utilizadas durante o treinamento podem ter degradado o desempenho geral do modelo e ao desaprender informações desatualizadas ou irrelevantes, os modelos de ML podem se tornar mais precisos.

4.3.2. Desafios no projeto de DM

Antes de conseguir efetivamente realizar a desaprendizagem de máquina, é necessário ultrapassar vários desafios para remover partes específicas dos dados de treino. Essas barreiras podem ser sintetizadas da seguinte forma:

- Estocasticidade do treinamento (stochasticity of training): Em virtude da natureza estocástica do procedimento de formação dos modelos de aprendizado de máquinas, não é possível saber a real influência que cada ponto de dados observado têm durante a formação no modelo [Bourtoule et al. 2021]. As redes neurais, por exemplo, são normalmente treinadas em mini-batches aleatórios que contêm um determinado número de amostras de dados. Além disso, o ordenamento dos lotes de formação também é aleatória [Bourtoule et al. 2021]. Esta característica estocástica gera dificuldades para a desaprendizagem de máquina, uma vez que a amostra de dados específica que precisa ser retirada teria de ser removida de todos os lotes. Logo, conhecimentos residuais ou vieses ainda podem permanecer no modelo

- Desaprendizado catastrófico (catastrophic unlearning): Em geral, um modelo desaprendido tem um desempenho pior do que o modelo retreinado do zero. No entanto, a degradação pode ser exponencial em relação a quantidade de dados a serem desaprendidos. Esta degradação súbita é frequentemente designada por desaprendizagem catastrófica [Nguyen et al. 2020]. Há inúmeros estudos que têm explorado formas de mitigar a desaprendizagem catastrófica através da definição de funções de perda e técnicas especiais para evitar este problema [Du et al. 2019, Golatkar et al. 2020a].

- Incrementalidade do treinamento (incrementality of training): O processo de treinamento de um modelo é um processo incremental, ou seja, a atualização do modelo numa determinada amostra de dados afetará o desempenho do modelo em amostras de dados introduzidas no modelo após essa inserção. O desempenho de um modelo nesta determinada amostra de dados também é afetado por amostras de dados anteriores. Determinar uma forma de eliminar o efeito da amostra de treinamento removida no desempenho do modelo é um desafio para a desaprendizado de máquinas [Nguyen et al. 2022].

- Padronização: Inexistência de métricas gerais para verificar na prática a eficiência e a eficácia do uso das técnicas de desaprendizado de máquinas.

4.4. Tipos de requisição de remoção

A remoção de informações de um modelo de aprendizagem de máquina é uma tarefa não trivial que exige a reversão parcial do processo de treinamento. Esta tarefa é inevitável

quando envolve dados sensíveis, como números de cartões de crédito ou palavras-passe, que acidentalmente entram no modelo e têm de ser removidos posteriormente. Dentre os principais tipos de requisição de remoção destacam-se:

- Remoção de pontos de dados (Item Removal): As solicitações de remoção de pontos de dados específicos são os casos mais comuns envolvendo a desaprendizado de máquina. Depois dos usuários partilharem os seus dados online, é geralmente difícil revogar o acesso à informação e pedir a eliminação dos seus dados.
- Remoção de atributos (Feature Removal): A desaprendizagem não deve limitar-se à remoção de pontos de dados, mas permitir correções em diferentes granularidades dos dados de treino, como a remoção de atributos. O primeiro método para desaprender atributos de um modelo foi proposto por [Warnecke et al. 2021]. A abordagem é inspirada no conceito de funções de influência, uma técnica estatística, que permite estimar a influência dos dados nos modelos de aprendizado [Koh and Liang 2017, Koh et al. 2019]. Ao reformular esta estimativa de influência como uma forma de desaprendizado, obtem-se uma abordagem versátil que mapeia as alterações dos dados de treino em retrospectiva com as atualizações dos parâmetros do modelo. Já o trabalho de [Guo et al. 2022] propõe uma nova abordagem para o desaprendizado de atributos associado a classificação por imagem. A Figura 4.4 apresenta a diferença inerente entre o atual paradigma de desaprendizagem de máquinas (esquerda) e a proposta de desaprendizagem de atributos (direita). A diferença reside principalmente nos objetos de remoção seletiva. A desaprendizagem de máquina existente visa remover seletivamente qualquer influência de certas amostras das representações de características aprendidas. Quando a desaprendizagem de máquinas existente remove amostras de entrada que contêm determinados atributos, o inconveniente é óbvio: para além dos atributos cuja remoção é solicitada, alguns atributos-chave que contribuem para o desempenho do modelo também foram removidos das representações de características aprendidas.

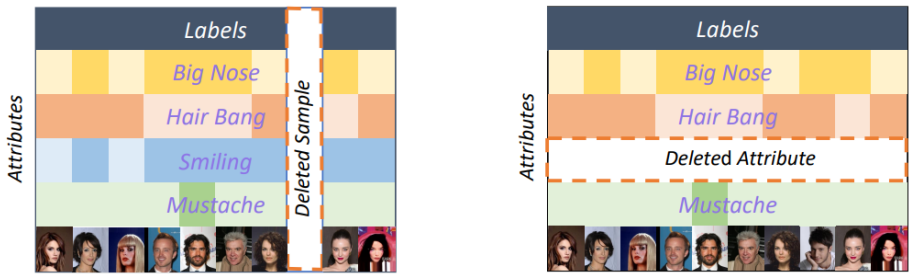


Figure 4.4. Comparação entre remoção por ponto de dados e por atributo

- Remoção de classes (Class Removal): Existem muitos cenários em que os dados a esquecer pertencem a uma ou várias classes de um modelo treinado. Por exemplo, nas aplicações de reconhecimento facial, cada classe é o rosto de uma pessoa, pelo que pode haver milhares ou milhões de classes. No entanto, quando um usuário opta por sair do sistema, a informação sobre o seu rosto deve ser removida sem utilizar uma amostra dele. Neste tipo de requisição a desaprendizagem exige que o

modelo esqueça uma ou mais classes, mas se lembre do resto dos dados. Para que uma ou mais classes sejam esquecidas, se o modelo puder ser atualizado através da observação de padrões que são, de alguma forma, opostos aos padrões aprendidos na altura do treino original, então os pesos atualizados do modelo podem refletir a desaprendizagem desejada e espera-se que preserve a informação sobre as restantes classes. O trabalho de [Tarun et al. 2023] propõe um método para remoção de classes baseado no chamado "data augmentation", que se caracteriza por introduzir ruído na base de treino com o objetivo de maximizar o erro na classe alvo para tratar deste tipo de remoção. Logo, é gerada uma matriz de ruído que é utilizada para manipular os pesos do modelo, com o intuito de desaprender a classe de dados alvo e manter o nível de performance do modelo original. Uma das suas principais vantagens está no fato de não ser necessário ter acesso as amostras de treinamento da classe alvo.

- Remoção de tarefas (Task Removal): Atualmente, os modelos de ML não são apenas treinados para uma única tarefa, mas também para várias tarefas. Este paradigma que busca espelhar o cérebro humano é denominado de aprendizado contínuo. No entanto, há situação em que pode ser necessário remover dados privados relacionados com uma tarefa específica. Por exemplo, considere um robô que foi treinado para assistir um doente em casa durante o seu tratamento médico. Pode ser pedido a este robô que esqueça este comportamento de assistência depois de o doente ter recuperado. Para este efeito, a aprendizagem de uma tarefa e o seu esquecimento no futuro tornou-se uma necessidade para os modelos de aprendizado contínuo.

- Remoção de fluxo (Stream Removal): No contexto da desaprendizado de máquinas, o tratamento de fluxos de dados está associada a lidar com um fluxo de pedidos de remoção. O trabalho de [Gupta et al. 2021] propôs um cenário de desaprendizagem em fluxo contínuo envolvendo uma sequência de pedidos de remoção de dados, motivado pelo fato de muitos usuários poderem estar envolvidos num sistema de ML e decidir apagar os seus dados sequencialmente. É também o caso quando os dados de treino foram envenenados por um ataque e os dados têm de ser eliminados gradualmente para recuperar o desempenho do modelo.

4.5. Métricas de avaliação

Esta seção busca apresentar métricas que avaliam o impacto do desaprendizado de máquinas no desempenho do modelo e que sejam capazes de quantificar o nível de esquecimento do processo de DM.

1. Acurácia: Um modelo desaprendido deve ser capaz de prever corretamente as amostras de teste ou, pelo menos, a sua acurácia deveria ser comparável à do modelo retreinado. No entanto, como o retreinamento é computacionalmente dispendioso, os modelos retreinados nem sempre estão disponíveis para comparação. Para resolver este problema, a acurácia do modelo não aprendido é frequentemente medida num novo conjunto de testes, ou é comparada com a do modelo original antes da desaprendizagem [He et al. 2021].

2. Completude (Completeness): Uma característica fundamental de um algoritmo de desaprendizagem é que ele deve ser completo. Esta métrica é capaz de avaliar se o esquecimento do sistema foi realmente integral, por meio da comparação entre os modelos desaprendidos e retreinados. Se o sistema atingiu a completude, os dois modelos farão as mesmas previsões sobre qualquer amostra de dados, estando elas corretas ou não. Para medir empiricamente a completude, quantifica-se a percentagem de amostras de entrada que recebem os mesmos resultados de previsão tanto do sistema desaprendido como do retreinado, utilizando um conjunto de dados de teste representativo. Quanto maior a percentagem, mais completa é a desaprendizagem [Cao and Yang 2015]. Há várias medidas de distância ou divergência que podem ser usadas para quantificar a diferença entre os modelos desaprendidos e retreinados, dentre as medidas representativas incluem a distância L2 e a divergência de Kullback-Leibler (KL) [Xu et al. 2024]. O trabalho de [Cao and Yang 2015] avalia analiticamente o uso desta métrica no processo de desaprendizagem de um algoritmo de recomendação chamado de LensKit. Para esquecer uma classificação, é necessário atualizar a sua matriz de semelhança de cada item. Os resultados para atualizar a semelhança entre os itens k e l são calculados do zero durante o retreinamento e comparados com uma medida conhecida chamada similaridade por cosseno, descrita na fórmula 1 abaixo, em que $||x||2$ representa a norma euclidiana de x, e $a_{*,k}$ é um vetor, (a_{1k}, a_{2k}, ..., a_{nk}), que representa todas as classificações recebidas pelo item k.

$$sim(k,l) = \frac{\overrightarrow{a_{*,k}}.\overrightarrow{a_{*,l}}}{||\overrightarrow{a_{*,k}}||_2||\overrightarrow{a_{*,l}}||} \tag{1}$$

3. Timeliness: Esta também é uma métrica importante para avaliar os sistemas que desaprenderam, pois ela mede o quão mais rápido é o desaprendizado para atualizar as features e o modelo do sistema em relação ao retreino. Ela está associada à rapidez do sistema em restaurar a privacidade, a segurança e a facilidade de utilização. Para medir empiricamente esta métrica, quantifica-se o aumento de velocidade da desaprendizagem em relação ao retreinamento. A desaprendizagem funciona melhor quando os dados a esquecer são pequenos em comparação com o conjunto de treino. Este caso é bastante comum. Por exemplo, os dados privados de um único usuário são normalmente pequenos quando comparados com os dados de treino de todos os usuários. Do mesmo modo, um atacante só precisa de uma pequena quantidade de dados para poluir um sistema de aprendizagem. Quando os dados a sereme esquecidos se tornam grandes, o retreino pode funcionar melhor [Cao and Yang 2015]. Além da completude, o trabalho de [Cao and Yang 2015] avalia o timeless associado ao processo de desaprendizagem do algoritmo LensKit, na qual constata-se que o nível de complexidade da desaprendizagem é $O(m^2)$, enquanto que para retreinamento é $O(nm^2)$. Logo, o algortimo de desaprendizagem tem uma performance melhor na ordem de $O(n)$ em comparação com o retreino.

4. Tempo de reaprendizagem (Relearn Time): Avalia a qualidade do desaprendizado em função do tempo de reaprendizado da informação removida. O tempo de reaprendizagem é um excelente indicador para medir a quantidade de informação de dados

não aprendidos que resta no modelo. Se um modelo recupera seu desempenho nos dados desaprendidos com apenas algumas etapas de retreinamento, é extremamente provável que o modelo tenha retido algum conhecimento dos dados desaprendidos.

5. Eficiência (Time Efficiency): A eficiência de um algoritmo de desaprendizagem é calculada empiricamente pelo razão entre o tempo necessário para obter o modelo desaprendido h^u e o tempo necessário para obter o modelo naıve retreinado h^*, conforme pode ser visto na fórmula 2 [Mercuri et al. 2022]. Uma eficiência alta indica um rápido desaprendizado.

$$Efficiency(h^u) = \frac{\text{time taken to obtain } h^*}{\text{time taken to obtain } h^u} \tag{2}$$

6. ZRF Score (Zero Retrain Forgetting): Esta métrica caracteriza-se por permitir a avaliação de métodos de desaprendizagem sem dependência do modelo retreinado, medindo a aleatoriedade das previsões do modelo, comparando a distribuição de saída do modelo desaprendido a um baseline aleatório [0,1], em que 1 indica modelo desaprendido apresenta comportamento aleatório e 0 exibe um certo padrão [Chundawat et al. 2023a]. A métrica ZRF score está descrita na fórmula 3, na qual é calculada utilizando divergência de Jensen-Shannon (JS) entre o modelo desaprendido M e um modelo inicializado aleatoriamente T_d, em relação à x_i que representa a i-ésima amostra do conjunto a esquecer D_f, notação apresentada anteriormente na Figura 4.3.

$$ZRF = 1 - (1/n_f)\sum_{i=0}^{n} JS(M(x_i), T_d(x_i)) \tag{3}$$

7. Anamnesis Index (AIN): Esta métrica foi introduzida por [Chundawat et al. 2023b], sob o pretexto de que apesar do tempo de reaprendizado ser bastante utilizado como métrica para avaliar a qualidade do desaprendizado ela possui algumas deficiências. Nas experiências realizada pelos autores, foi identificado que por vezes, os modelos desaprendidos recuperam uma acurácia significativa num número muito reduzido de passos de reaprendizagem, mas não convergem para a acurácia original na(s) classe(s) esquecida(s) durante um grande número de passos. Logo, foi proposto a métrica AIN, conforme pode ser visto na fórmula 4, que utiliza um valor percentual α em torno da acurácia original para calcular o tempo de reaprendizagem. Seja rt(M, M_{orig}, α) o número de passos necessários para que um modelo M se aproxime do percentual α da acurácia do modelo original M_{orig} nas classes esquecidas. Se M_u e M_s denotam o modelo não aprendido e o modelo treinado do zero. O valor de AIN varia de 0 a ∞, quanto mais próximo de 1 melhor é a desaprendizagem. Os valores de AIN muito inferiores a 1 correspondem aos casos em que a informação das classes esquecidas ainda está presente no modelo. Também indica que o modelo desaprendido reaprende rapidamente a fazer previsões exatas. Finalmente, caso a pontuação AIN for muito superior a 1, pode sugerir que a abordagem provoca alterações graves nos parâmetros.

$$AIN = \frac{rt(M_u, M_{orig}, \alpha)}{rt(M_s, M_{orig}, \alpha} \tag{4}$$

8. Distância de ativação (Activation Distance): É a separação entre a ativação final dos pesos eliminados e o modelo treinado novamente. Uma distância de ativação mais curta indica uma desaprendizagem melhor.

9. Epistemic Uncertainty: é uma métrica de incerteza que avalia se parâmetros atuais do modelo são ótimos para um determinado conjunto de dados. Com base nesta métrica, foi criado a efficacy score, conforme pode ser observado na fórmula 6, em que $i(w;D)$ determina a quantidade de informação que os parâmetros do modelo w consegue reter do dataset D. Esta pontuação mede a quantidade de informação que o modelo expõe. Quanto melhor for a estratégia de desaprendizagem, ela produzirá um modelo desaprendido com uma pontuação de eficácia mais baixa.

$$efficacy(w;D) = \begin{cases} \frac{1}{i(w;D)}, & \text{if } i(w;D) > 0 \\ \infty, & \text{se } n \text{ otherwise} \end{cases} \tag{5}$$

4.6. Requisitos de projeto

Para desenvolver um algoritmo de DM é necessário identificar alguns requisitos e definir a importância de cada um deles para o projeto. A figura 4.5 apresenta um diagrama contendo todos os requisitos, que estão descritos a seguir:

- Agnosticismo (Model-agnostic): Um processo de desaprendizagem deve ser genérico, ou seja, não estar condicionado a nenhum parâmetro ou especificações de qualquer algoritmo. Logo, ele deveria capaz de ser utilizado em diferentes modelos de aprendizagem de máquina [Bourtoule et al. 2021].

- Leveza (Light-weight): Outro fator fundamental para preparar o processo de desaprendizagem, muitas técnicas precisam armazenar pontos de controle do modelo, histórico de atualizações, dados de treinamento e outros dados temporários [He et al. 2021, Liu et al. 2020]. Um bom algoritmo de desaprendizagem deve ser leve e escalável com grandes volumes de dados. Ele deve ser capaz de lidar com sobrecarga computacional, para além do tempo de desaprendizagem e do custo de armazenamento [Bourtoule et al. 2021].

- Garantias comprováveis (Provable guarantees): É prático que um método de desaprendizagem forneça uma garantia comprovável sobre o modelo desaprendido. Para este efeito, muitos trabalhos conceberam técnicas de desaprendizagem com aproximações limitadas ao retreinamento, como trabalho [Guo et al. 2019] que criou uma garantia teórica muito forte de que um modelo do qual os dados são removidos não pode ser distinguido de um modelo que nunca observou os dados para começar. No entanto, abordagens com essa partem da premissa de que modelos com parâmetros comparáveis terão uma acurácia comparável.

- Capacidade de verificação (Verifiability): Para além dos pedidos de desaprendizagem, outra exigência dos usuários é verificar se o modelo desaprendido protege agora a sua privacidade. Para tal, um bom framework de desaprendizagem deve fornecer aos usuários finais um mecanismo de verificação. Por exemplo, os ataques backdoor podem ser utilizados para verificar a desaprendizagem, injetando amostras backdoor nos dados de treino [Sommer et al. 2020]. Se o backdoor não puder ser detectado no modelo original e for detectado no modelo desaprendido, a verificação é considerada um sucesso. No entanto, esta verificação pode ser demasiado intrusiva para um sistema de aprendizagem de máquina e a verificação pode ainda introduzir falsos positivos devido à incerteza inerente à detecção de backdoors.

- Completude (Completeness): Um bom algoritmo de desaprendizagem deve ser completo, ou seja, os resultados das previsões obtidas no modelo não aprendido e o retreinado devem ser consistentes sobre qualquer amostra de dados. Uma forma de medir esta consistência é calcular a percentagem dos mesmos resultados de previsão num dado de teste. Este requisito pode ser concebido como um objetivo de otimização numa definição de desaprendizagem, formulando a diferença entre o espaço de saída dos dois modelos. Há muitos trabalhos sobre ataques adversários podem ajudar nesta formulação [Chen et al. 2021, Sommer et al. 2022].

- Timeless: Outro importante requisito para bom algoritmo de desaprendizagem está associado a rapidez com ele desaprende. O timeless é utilizado para medir a rapidez da desaprendizagem em relação ao retreinamento após a solicitação de um pedido de desaprendizagem.

- Acurácia (Accuracy): Outra característica importante do algoritmo de desamprendizagem é a acurácia com que realiza as tarefas(previsão, classificação,...). Um modelo não aprendido deve ser capaz de prever corretamente as amostras de teste. A acurácia do modelo não aprendido é frequentemente medida num novo conjunto de testes, ou é comparada com a do modelo original antes da desaprendizagem.

4.6.1. Como os modelos “esquecem”?

Após um pedido de remoção de dados, o modelo atual será processado por um algoritmo de desaprendizagem para esquecer a informação correspondente a esses dados dentro do modelo. O algoritmo de desaprendizagem que permite o esquecimento seletivo, pode ter em conta vários requisitos, como a exaustividade, a atualidade e as garantias de privacidade. O resultado é um modelo não aprendido, que será avaliado em função de diferentes métricas de desempenho, descritas na seção 2.5. A Figura 4.6 apresenta a visão geral de como os modelos "esquecem".

4.7. Taxonomia e Algoritmos de DM

Dentre as possíveis classificações que as diferentes abordagens de desaprendizado de máquinas podem ter, este trabalho optou pela divisão entre os seguintes critérios:

1. Quanto ao grau de remoção de influência alcançado

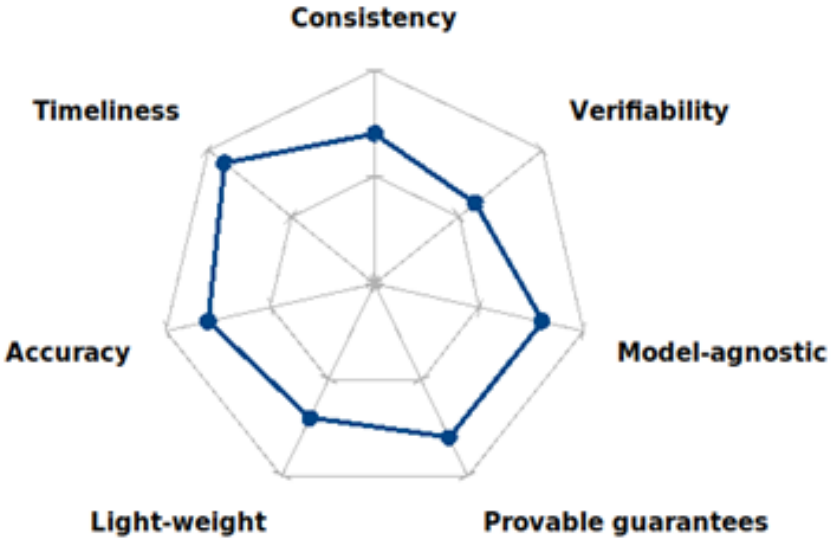


Figure 4.5. Requisitos de projeto para um algoritmo de desaprendizagem

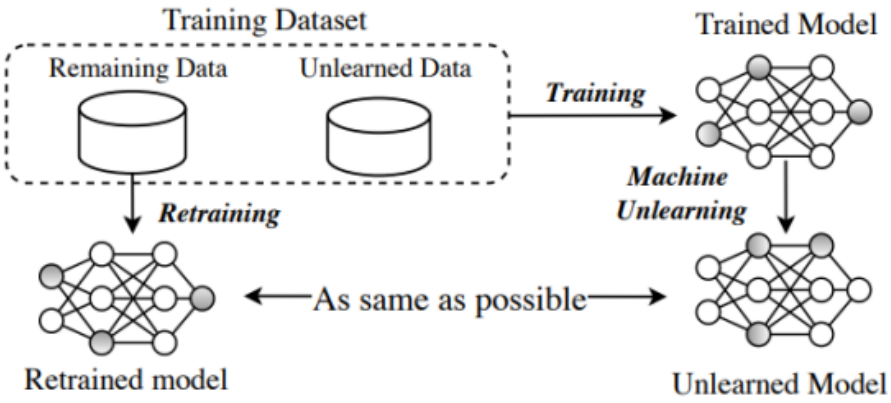


Figure 4.6. Visão geral de comos os modelos "esqucem"

- Exact Unlearning
- Approximate Unlearning

2. Quanto ao agnosticismo do algoritmo de aprendizado
   - Abordagens Model-agnostic
   - Abordagens Model-intrinsic
   - Abordagens Data-driven
3. Quanto à necessidade de dados de treinamento
   - Zero-Shot Unlearning

- One-Shot Unlearning
- Few-Shot Unlearning

4.7.1. Exact Unlearning

Exact unlearning é uma técnica que visa apagar completamente o impacto de dados específicos de treinamento em um modelo treinado. Quando um usuário solicita a exclusão de seus dados, a simples retirada dos mesmos não é suficiente, pois os modelos treinados ainda podem conter informações sensíveis e os dados retirados podem exercer influência sobre os parâmetros do modelo. Diante de tal problema, o exact unlearning garante que os dados que precisam ser desaprendidos sejam completamente removidos do modelo treinado, tornando o modelo desaprendido e o retreinado distribucionalmente idênticos. Em outras palavras, eles podem ser exatamente iguais diante de qualquer amostra de dados.

Matematicamente, o trabalho [Nguyen et al. 2022] apresenta a formulação geral desta abordagem, que permite definir o espaço métrico ao qual pertencem os modelos e consequentemente as suas distribuições. Dado um algoritmo de aprendizagem A(.), afirma-se que o processo U(.) é um processo de desaprendizagem exato se $\forall T \subseteq H, D \in Z^*, D_f \subseteq D$:

$$Pr(A(D|D_f) \in T) = Pr(U(D, D_f, A(D)) \in T) \tag{6}$$

Em termos das vantagens e desvantagens associado ao uso desta técnica, a seguir foram apresentas as principais características do desaprendizado:

- Remoção completa da influência de pontos de dados específicos (como se os dados removidos nunca tivessem sido vistos);
- Isola a influência de pontos exigindo apenas o retreinamento dos componentes afetados;
- Alto grau de consistência e verificabilidade;
- Necessário algum grau de retreinamento;
- Problema quanto a eficiência do retreinamento / Alto custo computacional;
- Induz a deterioração do desempenho;
- Sujeito a ataques de inferência/privacidade.

Portanto, em virtude dos inúmeros benefícios associados ao emprego desta abordagem, pesquisas têm surgido visando desenvolvimento de novos algoritmos que realizam o desaprendizado exato.

4.7.1.1. Exact Unlearning: SISA

Diante de dificuldade de lidar com um pedido de remoção de pontos de dados, o trabalho de [Bourtoule et al. 2021] apresentou o denominado treinamento SISA, sigla que indica:

- Sharded: particiona o dataset em conjuntos disjuntos (shards). Cada shard treina um submodelo
- Isolated: cada ponto de dados e sua influência está restrita a um único shard
- Sliced: adiciona mais um nível de fragmentação
- Aggregation: estratégia de votação majoritária

Este framework caracteriza-se por reduzir a sobrecarga computacional associada à desaprendizagem, mesmo no pior dos casos, em que os pedidos de desaprendizagem são efetuados uniformemente em todo o conjunto de treino. O treinamento SISA diminui o número de parâmetros do modelo afetados por um pedido de desaprendizagem e armazena em cache os resultados intermediários do algoritmo de treinamento para limitar o número de atualizações do modelo que têm de ser calculadas para que estes parâmetros sejam desaprendidos. Na Figura 4.7 é possível observar o funcionamento deste framework, em que os dados são divididos em shards, que por sua vez são divididos novamente em slices. Um modelo constituinte M é treinado em cada shard, apresentando-lhe um número crescente de slices e guardando os seus parâmetros antes do conjunto de treino ser aumentado com uma novo slice. Quando os dados precisam de ser desaprendidos, apenas um dos modelos constituintes cujos shards contém o ponto a desaprender precisa de ser treinado de novo - o treino pode começar a partir dos últimos valores de parâmetros guardados antes de incluir o shard que contém os pontos a serem desaprendidos.

A capacidade de generalização deste framework é notória, a ponto do trabalho de [Xu et al. 2024] definir o SISA como a abordagem geral para a desaprendizagem exata, dividindo as demais estruturas entre as que se baseiam no SISA ou não (non-SISA). O artigo separa as estruturas nos seguintes grupos:

- Exact Unlearning para Random Forest
- Exact Unlearning para modelos baseados em grafos
- Exact Unlearning para k-Means
- Exact Unlearning para Federated Learning
- Modelos não baseados no SISA (non-SISA)

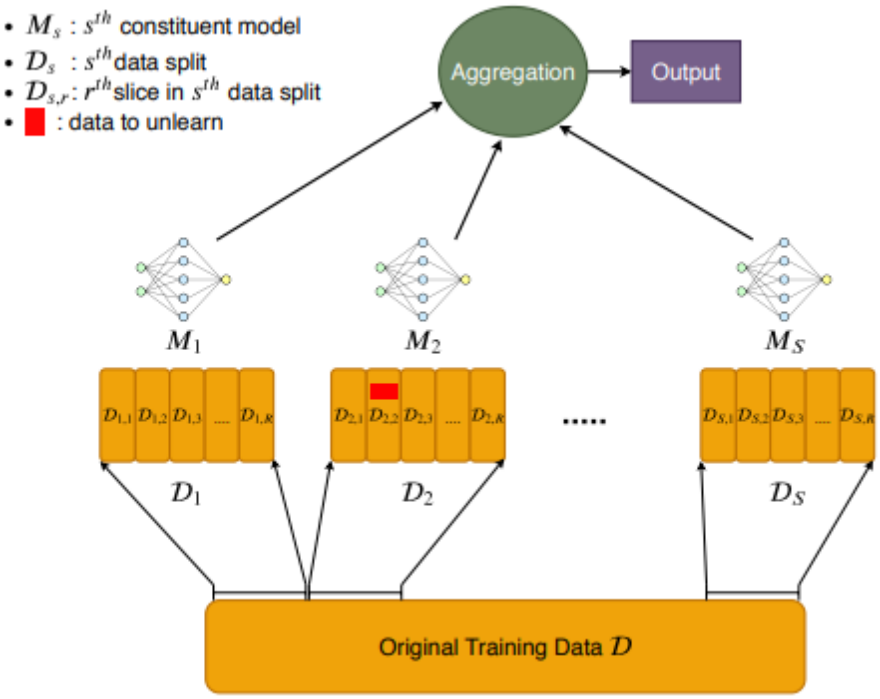


Figure 4.7. Treinamento SISA

4.7.1.2. Exact Unlearning para Random Forest

Cada árvore é treinada num subconjunto diferente de dados, atuando como um shard na estrutura SISA. As previsões das árvores individuais são agregadas para obter a previsão final da random forest. A influência de um ponto de dados é isolada dentro das árvores treinadas no subconjunto que contém esse ponto de dados. Ao desaprender um ponto de dados, apenas as árvores treinadas no subconjunto relevante necessitam de ser novamente treinadas

Dentre as abordagens para random forest, a proposta de [Brophy and Lowd 2021] permite a remoção de dados de treinamento com o mínimo de retreinamento. As atualizações do modelo para cada árvore DaRE na floresta são exatas, o que significa que a remoção de instâncias de um modelo DaRE produz exatamente o mesmo modelo que o retreino a partir do zero.

As árvores DaRE utilizam a aleatoriedade e o armazenamento em cache para tornar a eliminação de dados eficiente. Os níveis superiores das árvores DaRE utilizam nós aleatórios, que escolhem atributos de divisão e limiares uniformemente ao acaso. Estes nós raramente necessitam de atualizações porque dependem minimamente dos dados. Nos níveis inferiores, as divisões são escolhidas para otimizar avidamente um critério de divisão, como o índice de Gini ou a informação mútua. As árvores DaRE armazenam estatísticas em cada nó e dados de treino em cada folha, de modo a que apenas as subárvores necessárias sejam atualizadas à medida que os dados são removidos.

Portanto, as florestas DaRE utilizam várias técnicas para tornar as eliminações efi-

cientes (1) apenas retreinam partes do modelo em que a estrutura tem de mudar para corresponder à base de dados atualizada; (2) consideram no máximo k thresholds selecionados aleatoriamente por atributo; (3) introduzem nós aleatórios no topo de cada árvore que dependem minimamente dos dados e, portanto, raramente precisam ser retreinados. As Figuras 4.8 representa treinamento de uma árvore DaRE.

Algorithm 1 Building a DaRE tree / subtree.

Input: data D, depth d
if stopping criteria reached **then**
  $node \leftarrow$ LEAFNODE()
  save instance counts($node$, D) ▷ $|D|$, $|D_{\cdot,1}|$
  save leaf-instance pointers($node$, D)
  compute leaf value($node$)
else
  if $d < d_{\text{rmax}}$ **then**
    $node \leftarrow$ RANDOMNODE()
    save instance counts($node$, D) ▷ $|D|$, $|D_{\cdot,1}|$
    $a \leftarrow$ randomly sample attribute(D)
    $v \leftarrow$ randomly sample threshold $\in [a_{min}, a_{max})$
    save threshold statistics($node$, D, a, v)
  else
    $node \leftarrow$ GREEDYNODE()
    save instance counts($node$, D) ▷ $|D|$, $|D_{\cdot,1}|$
    $A \leftarrow$ randomly sample $\tilde{p}$ attributes(D)
    for $a \in A$ **do**
      $C \leftarrow$ get valid thresholds(D, a)
      $V \leftarrow$ randomly sample k valid thresholds(C)
      for $v \in V$ **do**
        save threshold statistics($node$, D, a, v)
    $scores \leftarrow$ compute split scores($node$)
    select optimal split($node$, $scores$)
  $D.\ell, D.r \leftarrow$ split on selected threshold($node$, D)
  $node.\ell$ = TRAIN(D_ℓ, $d+1$) ▷ Alg. 1
  $node.r \leftarrow$ TRAIN(D_r, $d+1$) ▷ Alg. 1
Return node

Figure 4.8. Treinamento de uma árvore DaRE

Os experimentos propostos pelos autores do DaRe, buscou medir a eficiência relativa comparando o tempo que um modelo DaRe elimina o número de instâncias com a abordagem de retreinamento naive. A avaliação da abordagem foi feita utilizando dois adversários diferentes: Random e Worst-of-1000. O adversário aleatório seleciona as instâncias de treino a serem eliminadas uniformemente ao acaso, enquanto o adversário pior-de-1000 seleciona cada instância, primeiro selecionando 1.000 instâncias candidatas uniformemente ao acaso e, em seguida, escolhendo a instância que resulta no maior retreino.

Tal como DaRe, HedgeCut [Schelter et al. 2021] é uma baseado no SISA e uma variante do random forest. Esta técnica se concentra em requisições de desaprendizado com baixa latência em árvores extremamente aleatórias. Ela introduz o conceito de ro-

bustez de divisão para identificar decisões de divisão que podem mudar com dados removidos. O HedgeCut mantém variantes de subárvores para esses casos e, ao desaprender um ponto de dados, substitui a divisão correspondente pelas variantes de subárvores preparadas. Esta operação é rápida e direta, garantindo um pequeno atraso na

4.7.1.3. Exact Unlearning para modelos baseados em grafos

A desaprendizagem exata de modelos baseados em grafos tem como objetivo remover de forma eficiente e precisa a influência de pontos de dados individuais nas previsões do modelo, tendo em conta as características únicas dos dados estruturados em grafos.

Dentre as principais abordagens está o GraphEraser [Chen et al. 2022], que expande a estrutura SISA a dados em grafos. O GraphEraser foi concebido para a desaprendizagem de redes neurais em grafos (GNNs). Ela consiste em três fases: divide o grafo de treino original em shards disjuntos, treina paralelamente um conjunto de modelos de shards F_i e aprende uma pontuação de importância ótima α_i para cada modelo de shard. Quando um nó w precisa de uma previsão, o GraphEraser envia w a todos os modelos de shards e obtém os posteriores correspondentes, que são depois agregados utilizando a pontuação de importância ótima α_i para fazer a previsão. Quando um nó u efetua um pedido de desaprendizagem, o GraphEraser remove u do shard correspondente e volta a treinar o modelo do shard.

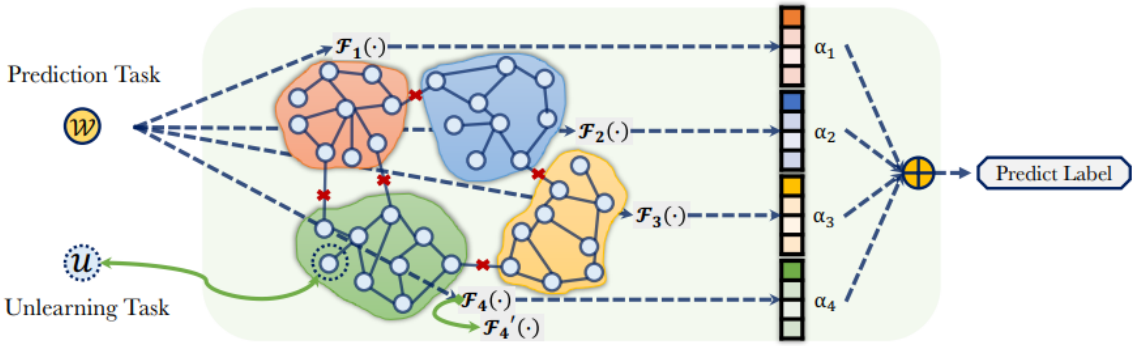


Figure 4.9. Framework do GraphEraser

4.7.1.4. Exact Unlearning para k-Means

Para a técnica baseada no k-means, tem-se a abordagem DC-k-means [Ginart et al. 2019], cuja sigla DC significa "dividir-e-conquistar" (Divide-and-Conquer). Esta técnica se caracteriza por ser baseada no SISA, mas que utiliza um método de agregação hierárquica em forma de árvore. Em linhas gerais, ele funciona através da partição do conjunto de dados em pequenos subproblemas, resolvendo cada subproblema como uma instância k-mean independente e fundindo recursivamente os resultados.

Os dados de treino são divididos aleatoriamente em vários subconjuntos, cada um representado por um nó folha numa árvore. Um modelo k-means é treinado em cada

subconjunto, com cada nó de folha correspondendo a um modelo k-means. O modelo final é uma agregação de todos os modelos k-means nos nós das folhas da árvore, obtida através da fusão recursiva dos resultados dos nós das folhas até à raiz. Para desaprender um ponto de dados, o nó folha relevante é localizado e o modelo k-means correspondente é atualizado para excluir esse ponto de dados. O modelo atualizado substitui então o modelo antigo no nó folha, e as alterações propagam-se pela árvore para atualizar o modelo agregado final. A Figura 4.10 apresenta o pseudocódigo desta abordagem.

Esta técnica utiliza a hierarquia da árvore para modular a dependência do cálculo em relação aos dados. Logo, no momento da eliminação dos dados, só é necessário recomputar os subproblemas de uma folha até à raiz, o que a possibilita suportar operações de eliminação rápidas [Ginart et al. 2019].

Algorithm 2 DC-k-means

Input: data matrix $D \in \mathbf{R}^{n \times d}$
Parameters: $k \in \mathbf{N}$, $T \in \mathbf{N}$, tree width $w \in \mathbf{N}$, tree height $h \in \mathbf{N}$
Initialize a w-ary tree of height h such that each node has a pointer to a dataset and centroids
for $i = 1$ **to** n **do**
  Select a leaf node uniformly at random
  node.dataset.add(D_i)
end for
for $l = h$ **down to** 0 **do**
  for each node in level l **do**
    $c \leftarrow$ k-means++(node.dataset,k,T)
    node.centroids $\leftarrow c$
    if $l > 0$ **then**
      node.parent.dataset.add(c)
    else
      save all nodes as metadata
      return c //model output
    end if
  end for
end for

Figure 4.10. Pseudocódigo DC-kmeans

4.7.1.5. Exact Unlearning para Federated Learning

Para este tipo de abordagem, o trabalho de [Su Ningxin 2023] desenvolveu o KNOT, que adota o framework da SISA para a desaprendizado federado assíncrona a nível do cliente

durante o treinamento. Um mecanismo de agregação em clusters divide os clientes em vários clusters. O servidor só efetua a agregação de modelos dentro de cada grupo, enquanto os diferentes grupos treinam de forma assíncrona. Quando um cliente pede para remover os seus dados, apenas os clientes dentro do mesmo cluster precisam de ser novamente treinados, enquanto os outros clusters não são afetados e podem continuar normalmente. Para obter uma atribuição ótima cliente-cluster, o KNOT efetua a formulação como um problema de minimização lexicográfica. O objetivo é minimizar a classificação de correspondência entre cada cliente e o cluster atribuído, considerando tanto a velocidade de treino como a semelhança do modelo. Este problema de otimização de inteiros pode ser resolvido eficientemente como um Programação Linear (LP) usando um solver LP.

A Figura 4.11 apresenta um exemplo de agregação assíncrona em clusters em que quatro clientes foram atribuídos a dois clusters. Se o cliente n.º 4 pedir para apagar os efeitos dos seus dados do servidor, apenas os clientes do agregado n.º 2 precisam de ser novamente treinados, enquanto os clientes do agregado n.º 1 podem prosseguir normalmente com o seu processo de formação FL.

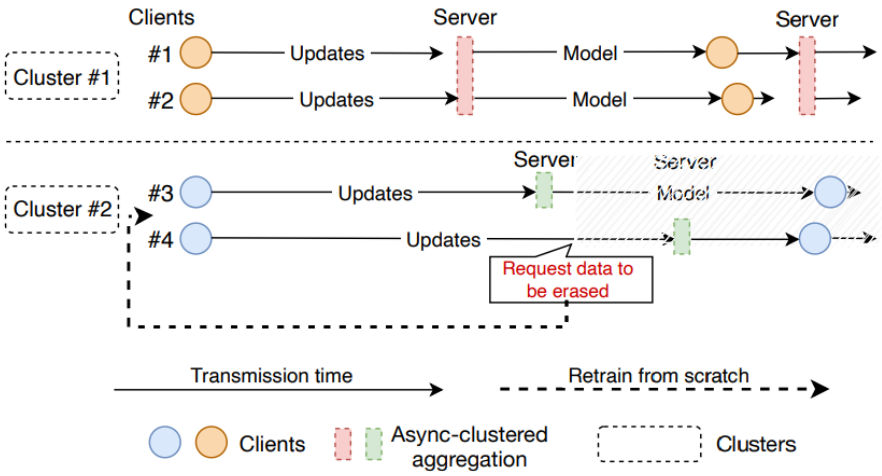


Figure 4.11. Exemplo de KNOT

Ao se analisar comparativamente as diferentes abordagenes de desaprendizado exato que baseiam no SISA, a Figura 4.12 apresenta o quadro geral contendo elementos como: objetivos, modelagem, pontos fortes e limitações de cada método.

4.7.1.6. Exact Unlearning: non-SISA

Dentre as abordagens que não se baseiam no SISA, o trabalho de [Cao and Yang 2015] concentra-se num dos desafios mais difíceis: fazer com que os sistemas de aprendizado de máquinas se esqueçam. A abordagem desenvolvida teve a seguinte ideia de desaprendizagem, em vez de fazer com que um modelo dependa diretamente de cada amostra de dados de treino (à esquerda), convertemos o algoritmo de aprendizagem numa forma de somatório (à direita) usando o conceito de SQL e Map-Reduce, o algoritmo de aprendizado

Paper	Goal	Design Ideas	Strengths	Weaknesses
SISA [36]	Efficient unlearning for general models	• Sharded, isolated, sliced, aggregated training	• Applicable to any model • Improves unlearning efficiency • Scalable	• Breaking data dependencies • Additional storage cost for model parameters • Accuracy degradation
DaRE [58]	Efficient unlearning for random forest	• Use cached statistics to only retrain affected subtrees • Consider subsets of thresholds per attribute	• Achieves efficiency by retraining affected subtrees • Analyzes the effect of layers and thresholds	• Additional storage cost for caching statistics • Worst case performance no better than retraining
HedgeCut [34]	Low-latency unlearning for ensemble of random forest	• Use split robustness to identify splitting decisions • Prepare subtree variants for non-robust splits	• Low-latency unlearning • Predictive accuracy similar to random forest	• Rely on the accuracy of predicted unlearning requests • Storage cost for subtree variants
DC-k-means [61]	Efficient unlearning for k-means	• Divide-and-Conquer approach • Hierarchical aggregation	• Theoretical guarantees on removal efficiency • Negligible quality loss	• State storage cost • High removing time complexity in high dimensions
GraphEraser [59]	Efficient unlearning for for GNNs	• Community detection for balanced sharding • Shard model importance scores	• Maintains graph structure • Balanced partitioning ensures unlearning efficiency	• Additional cost of maintaining massive shards. • Not satisfy the adaptive setting
RecEraser [60]	Efficient unlearning for recommender systems	• Different data partition strategies • Attention-based adaptive aggregation method	• Efficient unlearning with balanced data partition • Good global model utility with adaptive aggregation	• Specific to recommendation task • Open problem in the batch setting
KNOT [62]	Federated unlearning during training	• Cluster clients and perform aggregation within clusters • Clusters train asynchronously	• Limits retraining to a cluster • Asynchrony allows unaffected clusters to continue training	• Performance relies on cluster assignments • Asynchrony can negatively impact model accuracy
ARCANE [18]	Efficiently and accurately unlearning	• Multiple one-class classification tasks • Data preprocessing	• Maintains accuracy even with large unlearning requests • Data preprocessing accelerates unlearning	• Only applies to supervised models • Rely on preprocessing for efficiency

Figure 4.12. Visão geral dos métodos de desaprendizagem exata baseada no SISA [Xu et al. 2024]

é modificado em um sistema formado por somatórios. Especificamente, cada somatório é a soma de amostras de dados transformados, onde as funções de transformação g_i são eficientemente computáveis. Há apenas um pequeno número de somas e o algoritmo de aprendizado depende apenas delas. Para esquecer uma amostra de dados, basta atualizar os somatórios e depois calcular o modelo atualizado. Esta abordagem é muito mais rápida do que o retreinamento a partir do zero. O conceito desta abordagem pode ser visto na Figura 4.13.

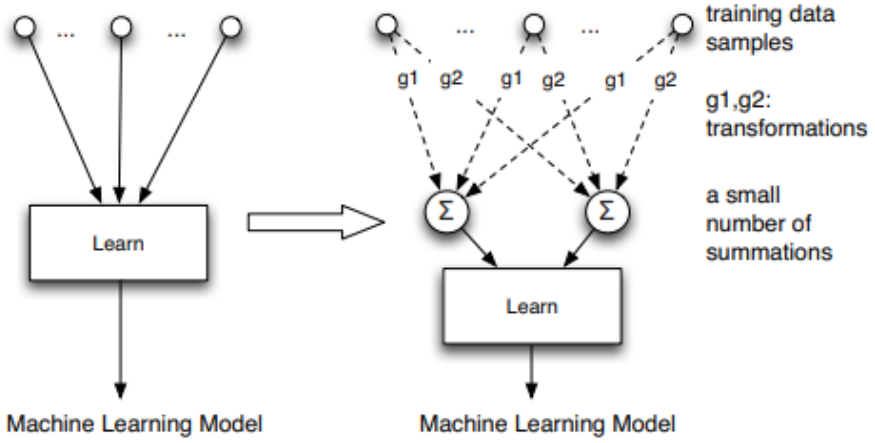


Figure 4.13. O conceito de desaprendizagem

Esta técnica é geral porque a forma de soma provém da aprendizagem da consulta estatística, do inglês "Statistical Query" (SQ) [48]. Muitos algoritmos de aprendizagem

de máquina, como os classificadores naıve Bayes, support vector machines e a técnica de clusterização k-means, podem ser implementados como aprendizado SQ. A nossa abordagem também se aplica a todas as fases da aprendizagem de máquina, incluindo a seleção de características e a modelização.

A abordagem de desaprendizado foi avaliada em quatro sistemas de aprendizado diferentes: o LensKit [Ekstrand et al. 2011], um sistema de recomendação de código aberto utilizado por vários sítios Web para recomendações de conferências e livros. O segundo foi uma reimplementação independente do Zozzle, detetor de malware JavaScript de código fechado cujo algoritmo foi adotado pelo Microsoft Bing. Já o terceiro foi um filtro de spam de redes sociais de código aberto [Gao et al. 2012]. Por último, teve também o PJScan, um detetor de malware PDF de código aberto [Laskov and Šrndić 2011]. Além disso, foi utilizado cargas de trabalho do mundo real, como mais de 100 mil amostras de malware JavaScript da Huawei.

System	Attack	Summation?	Analytical Results				Empirical Results				
			Complete-ness	Unlearn Speed	Retrain Speed	Speedup	Complete-ness	Unlearn Speed	Retrain Speed	Speedup	Modification LoC (%)
LensKit	Old	✓	100%	$O(m^2)$	$O(nm^2)$	$O(n)$	100%	45s	4min56s	6.57	302 (0.3%)
Zozzle	New	✓	100%	$O(q)$	$O(Nq)$	$O(N)$	100%	987ms	1day2hours	9.5×10^4	21 (0.4%)
OSNSF	New	✗	<100%	$O(logN)$	$O(NlogN)$	$O(N)$	99.4%	21.5ms	30mins	8.4×10^4	33 (0.8%)
PJScan	New	✓	100%	$O(N^p)$	$O(N^l)$	$O(N^k)$	100%	156ms	157ms	1	30 (0.07%)

Figure 4.14. Comparação dos resultados do desaprendizado

A Figura 4.14 apresenta o resumo desta abordagem de desaprendizado aplicado a todos os algoritmos de aprendizagem. Dentre os principais resultados, destaca-se o 100% de completude, um dos principais métricas para avaliação dos algoritmo de desaprendizado, para três sistemas LensKit, Zozzle e PJScan. Embora OSNF não tenha obtido tal performance, ele foi o sistema que desaprendeu mais rápido com 21.5 ms.

4.7.2. Approximate Machine Unlearning

Embora os métodos de desaprendizado de máquinas exata tenham dado passos significativos no sentido de alcançar o direito ao esquecimento, pesquisadores identificaram novos desafios a partir de duas outras perspectivas. A primeira é que o desaprendizado de máquinas exato resulta inevitavelmente numa degradação do desempenho do modelo, mesmo que essa degradação seja, por vezes, menor. O segundo é o potencial de fuga de privacidade. Por exemplo, se a "Alice" for removida e o modelo bruto diferir do modelo recalibrado, um adversário pode perceber que a diferença se deve à remoção da "Alice" e lançar um ataque de inferência para comprometer ainda mais a privacidade. Os dois desafios acima referidos podem ser atenuados através da utilização de métodos de desaprendizado de máquinas aproximados, que ocultam a diferença entre os modelos antes e depois da remoção dos dados, otimizando simultaneamente o desempenho do modelo [Qu et al. 2023].

Logo, o desaprendizado aproximada é um processo que tem por objetivo minimizar a influência dos dados não aprendidos para um nível aceitável, em vez de os eliminar completamente. Para isso ele envolve as seguintes etapas:

- Cálculo da influência: Calcular a influência dos pontos de dados que precisam de ser desaprendidos no modelo original. Isto implica determinar como estes pontos de dados afetam o modelo

- Ajuste dos parâmetros do modelo: Modificar os parâmetros do modelo para inverter a influência dos dados que estão a ser removidos. Este ajustamento envolve normalmente métodos como a reponderação ou o recálculo de parâmetros ótimos e a modificação do modelo para que se comporte como se tivesse sido treinado no conjunto de dados sem os pontos de dados não aprendidos.

- Adição de ruído (opcional): É adicionado ruído cuidadosamente calibrado para evitar que os dados removidos sejam inferidos a partir do modelo atualizado. Este passo assegura a confidencialidade do conjunto de dados de treino.

- Validação do modelo atualizado: Avaliar o desempenho do modelo atualizado para garantir a sua eficácia. Esta etapa de validação pode envolver uma validação cruzada ou um teste num conjunto de espera para avaliar a exatidão e a generalização do modelo.

Em virtude do desaprendizado aproximado ser uma abordagem que constitui uma alternativa prática à desaprendizagem exata, particularmente em cenários em que fatores como o custo computacional, o custo de armazenamento e a flexibilidade sejam cruciais [Qu et al. 2023]. As seguintes subseções apresentaram diferentes métodos que utilizam essa abordagem, com suas respectivas características e limitações.

4.7.2.1. Approximate Unlearning baseada na função de influência dos dados removidos

A ideia central deste tipo de abordagem consiste em utilizar a função de influência para estimar o nível de impacto que a remoção de pontos de dados têm sobre o modelo.

O primeiro trabalho envolvendo a abordagem foi o de [Guo et al. 2019], que propôs um método que ajusta os parâmetros do modelo com base na influência calculada dos dados removidos.envolve a aplicação de regularização nos parâmetros do modelo, como os coeficientes em modelos lineares, para reduzir a influência de dados específicos. O trabalho obtém uma remoção certificada de modelos lineares L2-regularizados, que envolve a aplicação de regularização nos parâmetros do modelo, como os coeficientes em modelos lineares, para reduzir a influência de dados específicos. Entretanto, a proposta apresentou limitações para lidar com modelos de perdas não convexas, porque a função de influência se basear no pressuposto de convexidade para garantir que a influência estimada reflete com precisão a verdadeira influência dos pontos removidos.

Com base no trabalho de Guo, [Sekhari et al. 2021] não requer acesso total ao conjunto de dados de treinamento durante o processo de desaprendizagem. Eles permitem um desaprendizado eficiente sem a necessidade de todos os dados de treinamento, reduzindo os requisitos computacionais e de armazenamento. A Figura 4.15 apresenta o pseudocódigo da proposta, em que A_{sc} consiste no algoritmo de aprendizado, recebe

como entrada o conjunto de pedidos de eliminação U, o ponto w e as estatísticas de dados T(S). Utilizando estes dados, A_{sc} começa por estimar a matriz H que denota o Hessiano da função empírica no conjunto de dados S/U quando avaliada no ponto wb. Em seguida, A_{sc} calcula o ponto w removendo a contribuição dos pontos U eliminados de wb utilizando a atualização em (8). Finalmente, A_sc perturba w com ruído retirado de $N(0, \sigma^2 I_d)$ e devolve o ponto perturbado w.

Algorithm 1 Unlearning algorithm ($\bar{A}_{sc}$)

Input: Delete requests: $U = \{z_j\}_{j=1}^{m} \subseteq S$, output of $A_{sc}(S)$: $\widehat{w}$, additional statistic $T(S)$: $\{\nabla^2 \widehat{F}(\widehat{w})\}$, loss function: $f(w, z)$.

1: Set $\gamma = \frac{2Mm^2L^2}{\lambda^3 n^2}$, $\sigma = \frac{\gamma}{\varepsilon}\sqrt{2\ln(1.25/\delta)}$.

2: Compute

$$\widehat{H} = \frac{1}{n-m}\left(n\nabla^2 \widehat{F}(\widehat{w}) - \sum_{z\in U} \nabla^2 f(\widehat{w}, z)\right). \tag{7}$$

3: Define

$$\bar{w} = \widehat{w} + \frac{1}{n-m}(\widehat{H})^{-1} \sum_{z\in U} \nabla f(\widehat{w}, u). \tag{8}$$

4: Sample $\nu \in \mathbb{R}^d$ from $\mathcal{N}(0, \sigma^2 \mathbb{1}_d)$.

5: **Return** $\widetilde{w} := \bar{w} + \nu$.

Figure 4.15. Pseudocódigo desaprendizado aproximado [Sekhari et al. 2021]

Outro trabalho que merece destaque foi desenvolvido por [Mehta et al. 2022], que aborda desaprendizado aproximado especificamente para redes neurais profundas (DNN). É proposto um esquema de seleção, L-CODEC, que identifica um subconjunto de parâmetros a atualizar, eliminando a necessidade de inverter uma grande matriz. Isto evita a atualização de todos os parâmetros, concentrando-se apenas nos mais influentes. Com base nisto, propõem L-FOCI para construir um conjunto mínimo de parâmetros influentes utilizando L-CODEC de forma incremental. Uma vez identificado o subconjunto de parâmetros a atualizar, aplicam uma atualização de Newton no sentido dos ponteiros do relógio a esse subconjunto. Ao concentrar os cálculos apenas nos parâmetros influentes, a sua abordagem torna viável a desaprendizagem aproximada para grandes redes neurais profundas, anteriormente inviáveis.

Na Figura 4.16 apresenta o exemplo do funcionamento quando uma amostra é perturbada e passada através da rede. As ativações são agregadas juntamente com as perdas e alimentam o L-FOCI. As linhas seleccionadas representam fatias da camada correspondente que são suficientes para a desaprendizagem.

Já dentre as abordagens diferentes de [Guo et al. 2019], que apenas ajusta a camada de decisão linear de um modelo, o trabalho proposto por [Wu et al. 2022] desenvolveu o PUMA, que caracteriza-se por modificar todos os parâmetros de treinamento, oferecendo uma solução mais completa para a remoção de dados. O principal objetivo do PUMA é manter o desempenho do modelo após a remoção dos dados, em vez de apenas monitorar se o modelo modificado consegue produzir previsões semelhantes às de um

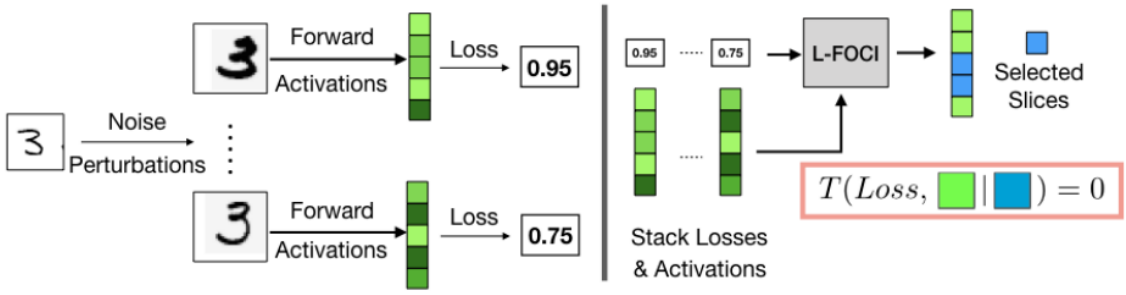


Figure 4.16. Exemplo de funcionamento L-FOCI

modelo treinado com os dados restantes, como faz o método de Guo et al. Para o conseguir, o PUMA utiliza a influência para medir o impacto que de cada ponto de dados tem no desempenho do modelo e, em seguida, ajusta o peso dos dados restantes para compensar a remoção de um ponto de dados específico. A Figura 4.17 apresenta o resultado da remoção dos pontos de treino marcados por cruzes do modelo. Como demonstrado no gráfico da direita, o PUMA removeu com sucesso a informação de todos os pontos marcados.

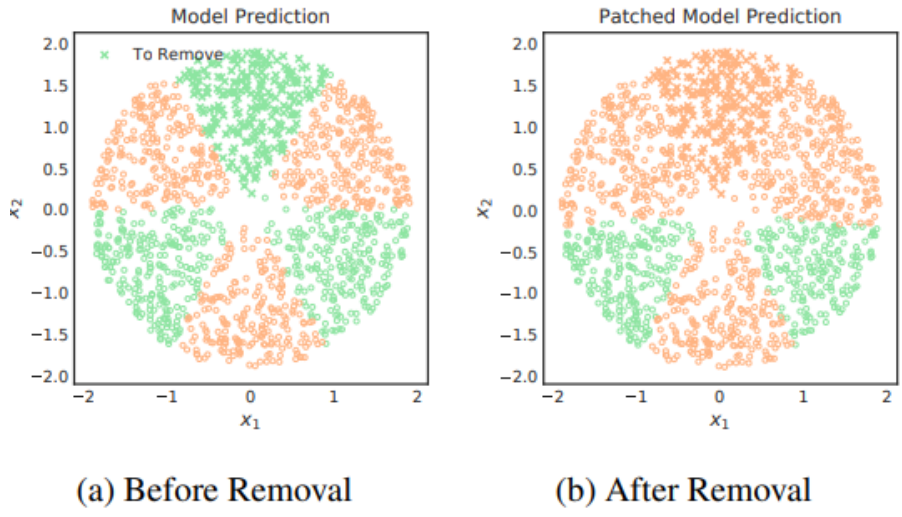


Figure 4.17. Removendo pontos dos dados usando PUMA

A Figura 4.18 apresenta o quadro geral das diferentes abordagens de desaprendizagem aproximada baseada na função de influência dos dados removidos, contendo características como: objetivos, modelagem, pontos fortes e limitações de cada método.

4.7.2.2. Approximate Unlearning baseada na reotimização após a remoção dos dados

A ideia central da desaprendizagem aproximada baseada na reotimização é ajustar iterativamente um modelo para esquecer efetivamente pontos de dados específicos, man-

Paper	Goal	Design Ideas	Strengths	Weaknesses
Guo *et al.* (2020) [43]	Certified removal	• One-step Newton update/influence function • Difference privacy	• Efficient training data removal • Strong theoretical certified removal guarantee	• Relies on convexity • High computational cost for inverting the Hessian matrix
Sekhari *et al.* (2021) [64]	Efficient unlearning with generalization guarantees	• Use influence functions to identify important data points • Store cheap data statistics	• Considers test loss instead of just training loss • Reduce computational and storage costs • Give the deletion capacity	• Relies on convexity • Relies on storage of data statistics • Does not handle finite/discrete hypothesis classes
Suriyakumar *et al.* (2022) [65]	Efficient unlearning for ERM models	• Infinitesimal jackknife • Newton update	• Computationally efficient online unlearning • Accommodating non-smooth regularizers	• Specific to ERM models • Inefficient for batch removal
Mehta *et al.* (2022) [66]	Efficient unlearning for DNN	• Conditional independence-based parameter selection	• Avoids full Hessian inverse • Improves unlearning efficiency in DNN	• Relies on weighted sampling based on Lipschitz constants of filters/layers • Dependence on newly developed optimization tools
PUMA [67]	Maintain performance during unlearning	• Performance unchanged model augmentation	• Maintain performance after removal • Computationally efficient	• Limited evaluation on simple datasets • Sensitive to hyperparameters
Tanno *et al.* (2022) [68]	Repair model by data removal	• Identify detrimental data via influence function • Remove via posterior approximation	• Model-agnostic framework • Identify causes of failures	• Limited to detrimental data removal • Cause identification can be computationally intensive
Warnecke *et al.* (2023) [70]	Unlearn features and labels	• Use influence functions as updates for features or labels	• Effective unlearning of features/labels • Efficient closed-form updates	• Efficacy drops as affected features/labels increase • No guarantee for non-convex models

Figure 4.18. Visão geral dos métodos de Desaprendizagem aproximada baseada na função de influência dos dados removidos [Xu et al. 2024]

tendo o desempenho global. Pesquisas neste domínio propõem diferentes técnicas de remoção/esquecimento seletivo com base nos objetivos da aplicação.

O primeiro trabalho sobre o tema foi feito por [Golatkar et al. 2020a], que propõem um algoritmo ótimo de depuração quadrática para conseguir o esquecimento seletivo em redes profundas. O esquecimento seletivo é definido como um processo que modifica os pesos da rede usando uma função de depuração S(w) para tornar a distribuição indistinguível dos pesos de uma rede nunca treinada com os dados esquecidos. O esquecimento seletivo é medido pela divergência KL, que dado p(x) e q(x) duas distribuições, essa métrica pode ser representado pela fórmula 7 .

$$KL(p(x)||q(x)) := E_{x\sim p(x)}[log(p(x)/q(x))] \tag{7}$$

Se a divergência KL entre a distribuição de pesos da rede após o esquecimento seletivo e a distribuição de pesos da rede que nunca viu os dados esquecidos for zero, as duas distribuições são exatamente iguais, o que indica um esquecimento completo (completeness). Logo, ela é capaz de medir a quantidade máxima de informação que um atacante pode extrair.

No seu trabalho de seguinte, [Golatkar et al. 2020b] observam que as alterações de peso podem não afetar os resultados das redes profundas devido à sobreparametrização. Consequentemente, os atacantes ainda podem extrair dados removidos D_f das saídas. Para resolver esse problema, eles se concentram nas ativações finais. Essas ativações representam a resposta do modelo aos dados de entrada e refletem mais diretamente os

processos de memória e remoção. Para isso, eles utilizam um Neural Tangent Kernel (NTK) para correlacionar pesos e activações e introduzem um processo de depuração baseado em NTK para conseguir a remoção, minimizando a diferença entre a ativação da rede no conjunto de dados de remoção e o modelo alvo.

Shibata et al. apresentam a Aprendizagem com Esquecimento Seletivo (LSF) [Shibata et al. 2021] para conseguir o esquecimento seletivo ao nível da classe na aprendizagem contínuo (lifelong learning). Os autores introduzem uma função de perda F com quatro componentes: perda de classificação F_C para garantir a precisão da classificação, perda mnemónica F_M que associa cada classe a um código incorporado, perda de esquecimento seletivo F_{SF} para remover classes marcadas para remoção e perda de regularização F_R para evitar o esquecimento catastrófico, mencionado como um dos desafios do DM presentes na seção 2.3.

Os códigos mnemónicos permitem o esquecimento seletivo de qualquer classe, eliminando o seu código sem os dados originais. O modelo pode esquecer seletivamente certas classes através de uma nova otimização da perda F, como ilustrado na fórmula 8.

$$F = F_C + F_M + F_{SF} + F_R \tag{8}$$

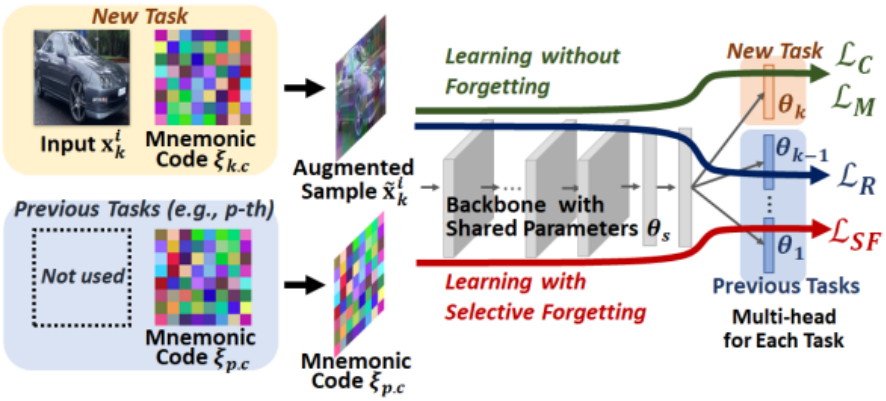


Figure 4.19. Visão geral do método LSF

A Figura 4.20 apresenta o quadro geral das diferentes abordagens de desaprendizagem aproximada baseada na reotimização após a remoção dos dados, contendo características como: objetivos, modelagem, pontos fortes e limitações de cada método.

4.7.2.3. Approximate Unlearning baseada na atualização do gradiente

A desaprendizagem aproximada baseada em atualizações de gradiente faz pequenos ajustes aos parâmetros do modelo para modificá-lo depois de remover ou adicionar incrementalmente pontos de dados. Esses métodos geralmente seguem uma estrutura de duas etapas para atualizar modelos treinados após pequenas alterações de dados sem retreinamento completo: a primeira consiste em inicializar os parâmetros do modelo usando o modelo

Paper	Goal	Design Ideas	Strengths	Weaknesses
Golatkar *et al.* (2020a) [41]	Selective forgetting in deep networks	• Optimal quadratic scrubbing on weights • Add noise tailored to the loss landscape	• Formal definition of selective forgetting • Provides upper bound on remaining info	• Rely on the stability of SGD • Computationally expensive, limited scalability
Golatkar *et al.* (2020b) [72]	Selective forgetting from input-output observations	• Analysis based on final activations • NTK-based scrubbing	• Provides tighter black-box bounds with limited queries • Handles over-parameterized models	• Relies on linearization assumptions • Computationally expensive
Golatkar *et al.* (2021) [73]	Selective forgetting for large deep networks	• Mixed-privacy setting • User weights obtained by minimizing quadratic loss	• Maintaining accuracy on large vision tasks • Provides information bounds	• Relies on the strong convexity assumptions • Forgetting quality depends on data subsets
Shibata *et al.* (2021) [74]	Class-level selective forgetting in lifelong learning	• Mnemonic codes • New loss function with four components	• Class-level removing • No need for original data	• Customized for image data • Computational cost of embedding codes

Figure 4.20. Visão geral dos métodos de desaprendizagem aproximada baseada na reotimização após a remoção dos dados [Xu et al. 2024]

treinado anteriormente. Já o segunda consiste em executar algumas etapas de atualização de gradiente nos novos dados.

Dentre os representantes desta categoria, é possível citar o Deltagrad [Wu et al. 2020] caracterizado por se adaptar os modelos de forma eficiente a pequenas alterações no conjunto de treino, utilizando o gradiente armazenado em cache e a informação sobre os parâmetros durante o processo de treino original. O algoritmo inclui dois casos: iteração de burn-in e outras iterações.

Outra abordagem é chamada de Descent-to-Delete, proposta por [Neel et al. 2021] introduz um algoritmo básico de descida de gradiente que começa com o modelo anterior e executa um número limitado de atualizações de descida de gradiente. Este processo assegura que os parâmetros do modelo permanecem muito próximos dos parâmetros ótimos. O ruído gaussiano é aplicado aos parâmetros do modelo para garantir a indistinguibilidade de qualquer entidade próxima do modelo ótimo. Para dados de elevada dimensão, divide os dados e otimiza independentemente cada partição. A Figura 4.21 consiste no pseudocódigo do processo de desaprendizado do Descent-to-Delete.

Algorithm 2 $\mathcal{R}_{\mathcal{A}}$: ith **Unlearning** for Perturbed Gradient Descent

Input: dataset $\mathcal{D}_{i-1}$, update u_i, model θ_i
Update dataset $\mathcal{D}_i = \mathcal{D}_{i-1} \circ u_i$
Initialize $\theta'_0 = \theta_i$
for $t = 1, 2, \ldots T_i$ **do**
  $\theta'_t = \text{Proj}_\Theta\left(\theta'_{t-1} - \eta_t \nabla f_{\mathcal{D}_i}(\theta'_{t-1})\right)$
Output: $\hat{\theta}_i = \theta'_{T_i}$; // Secret output

Figure 4.21. Pseudocódigo Descent-to-Delete

A Figura 4.22 apresenta o quadro geral das diferentes abordagens de desaprendizagem aproximada baseada na função de influência dos dados removidos, contendo características como: objetivos, modelagem, pontos fortes e limitações de cada método.

Paper	Goal	Design Ideas	Strengths	Weaknesses
DeltaGrad [40]	Efficiently retrain models after minor data changes	• Cached training parameters and gradients • Burn-in iterations + L-BFGS approximation	• Applicable to general models with GD/SGD • Support both addition and removal	• Gradient storage cost • Relies on strong convexity assumptions
FedRecover [75]	FL model recovery after poisoning attacks	• Server estimates updates • L-BFGS approximation +corrections	• Estimate clients' model updates to reduce communication cost • Scalable to numerous clients	• Storing historical information • Convexity assumptions
Descent-to -Delete [76]	Unlearning with efficiency and privacy guarantees	• Gradient descent perturbations • Data partitioning	• Gaussian noise for indistinguishability • Handles arbitrary updates • Improved accuracy for high-dimensional data	• Convexity assumptions • Accuracy/efficiency tradeoff
BAERASER [77]	Remove backdoor	• Trigger pattern recovery • Gradient ascent unlearning	• Removes backdoors without retraining data • Prevents catastrophic forgetting	• Recovered triggers not identical • Applicable to backdoor only

Figure 4.22. Visão geral dos métodos de desaprendizagem aproximada baseada na reotimização após a remoção dos dados [Xu et al. 2024]

4.7.2.4. Approximate Unlearning em grafos

Os dados estruturados em grafos colocam desafios únicos à desaprendizagem de máquinas devido às dependências inerentes entre pontos de dados ligados. Os métodos tradicionais de DM concebidos para dados independentes não têm frequentemente em conta as interações complexas presentes nos dados de grafos.

Em primeiro lugar, a interdependência dos dados é um desafio fundamental na desaprendizagem de grafos. Dado um nó num grafo como alvo de remoção, é necessário remover a sua influência e a sua potencial influência nos vizinhos multi-hop. Para resolver este problema, [Wu et al. 2023] propuseram uma Função de Influência de Grafo (GIF) para considerar essa influência estrutural do nó/aresta/caraterística nos seus vizinhos. A GIF estima as alterações dos parâmetros em resposta a perturbações de massa nos dados removidos, introduzindo um termo de perda adicional relacionado com os vizinhos afetados. O GIF fornece uma forma de explicar os efeitos da desaprendizagem das características dos nós.

Outro trabalho que aborda o problema da interdependência dos dados em grafos, [Cheng et al. 2023] propôs o GNNDELETE, um método que integra um novo operador de eliminação para lidar com o impacto da eliminação de arestas em grafos. Eles introduziram duas propriedades chave, nomeadamente a consistência da aresta eliminada e a influência da vizinhança, para limitar o impacto da eliminação de arestas apenas à vizinhança local. A consistência da aresta eliminada garante que a eliminação de uma aresta não afeta a representação de outras arestas na mesma vizinhança. A influência da vizinhança garante que a eliminação de uma aresta afeta apenas os seus vizinhos directos e não todo o grafo.

A Figura 4.23 apresenta a visão geral do funcionamento do método. Dado um modelo GNN treinado e um pedido de eliminação de arestas, o GNNDELETE produz representações não aprendidas de forma eficiente, aprendendo apenas um pequeno operador de eliminação W_D. Ele também garante a qualidade da representação minimizando uma função de perda que satisfaz as duas propriedades-chave propostas no trabalho.

Um segundo problema envolvendo grafos, é que a maioria dos métodos de de-

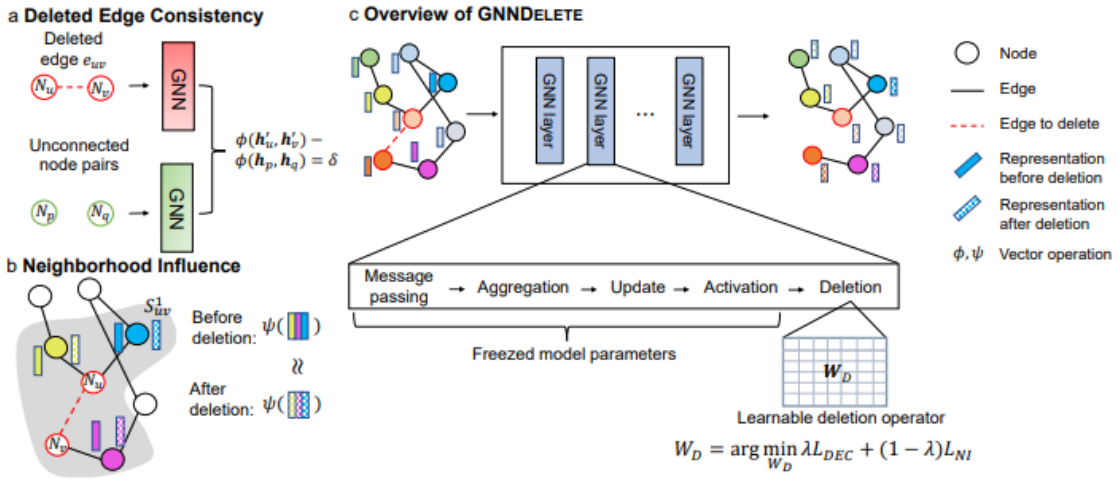


Figure 4.23. Visão geral do GNNDELETE

saprendizagem de grafos foi concebida para o cenário de aprendizado transdutivos, em que o grafo é estático e a informação do grafo de teste está disponível durante o treino. No entanto, muitos grafos do mundo real são dinâmicos, adicionando continuamente novos nós e arestas. Para resolver este problema, [Wang et al. 2023] propuseram o GUIded InDuctivE Graph Unlearning framework (GUIDE) para efetuar a desaprendizagem de grafos dinâmicos. O GUIDE inclui um particionamento justo e equilibrado dos grafos guiados, uma reparação eficiente dos subgrafos e uma agregação baseada na similaridade. O particionamento equilibrado garante que o tempo de reciclagem de cada fragmento seja semelhante, e o reparo de subgrafos e a agregação baseada em similaridade reduzem os efeitos colaterais do particionamento de grafos, melhorando assim a utilidade do modelo. A Figura 4.24 apresenta a visão geral do framework GUIDE.

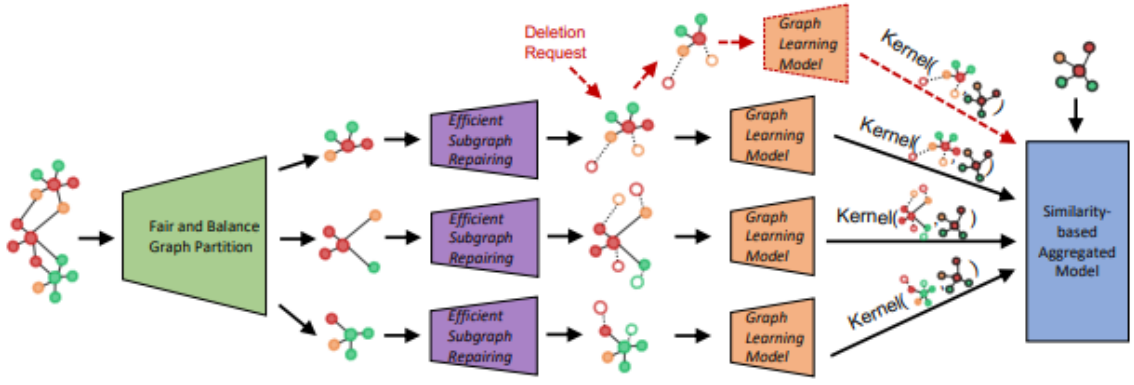


Figure 4.24. Visão geral do GUIDE

Em terceiro lugar, é mais difícil conseguir a desaprendizagem de grafos mantendo o desempenho do modelo quando o número de dados de treino é limitado. Para resolver este problema, [Pan et al. 2023] propuseram um método aproximado não linear de desaprendizagem de grafos baseado na Transformada de Dispersão de Grafos (GST). A GST é estável sob pequenas perturbações nas características e topologias dos grafos,

tornando-a um método robusto para o processamento de dados de grafos. Além disso, as GSTs não são treináveis e todos os coeficientes wavelet nas GSTs são construídos analiticamente, tornando as GSTs computacionalmente mais eficientes e exigindo menos dados de treino do que as GNNs

Há também abordagens no domínio da aprendizagem de embeddings de grafos com conhecimento federados (KG). Para esta área, uma preocupação importante é o tratamento eficaz da heterogeneidade dos dados e dos desafios da retenção de conhecimentos. A FedLU [Zhu et al. 2023] adota um processo de esquecimento em duas fases, de interferência e decaimento. Na primeira etapa de interferência retroativa, o FedLU efectua confusões duras e suaves, uma vez que a teoria da interferência afirma que o esquecimento ocorre quando as memórias competem e interferem com outras memórias [Wixted 2021].

A Figura 4.25 apresenta o quadro geral das diferentes abordagens de desaprendizagem aproximada baseada na função de influência dos dados removidos, contendo características como: objetivos, modelagem, pontos fortes e limitações de cada método.

Paper	Goal	Design Ideas	Strengths	Weaknesses
GIF [81]	General unlearning strategy for GNNs	Graph influence functions considering neighbors' influence	• Applicable to different models • Supports various removal tasks • Closed-form solution	• Focus on the classification • Memory-intensive
GNNDelete [82]	Efficient unlearning for GNNs	• Deleted Edge Consistency • Neighborhood Influence	• Flexible removal operator applicable to any GNN • Supports various removal tasks	• Limited to transductive learning
Chien *et al.* (2022) [83]	Graph-structured data unlearning	• Update model parameters based on gradient difference	• Analyze for node/edge/feature unlearning • Strong theoretical guarantees	• Analysis limited to linear models • Loose worst-case bounds
GUIDE [84]	Inductive graph unlearning	• Guided graph partitioning • Subgraph repairing • Similarity-based aggregation	• Repair subgraphs independently • Enables unlearning on evolving graphs	• Increased memory cost • Generalizing partition fairness needs exploration
Pan*et al.* (2023) [85]	Unlearning for GNNs with limited training data	Use GSTs for graph embeddings	• Nonlinear approximate graph unlearning • Theoretical guarantees	• Limited to classification • Bounds are loose
FedLU [86]	Unlearning for knowledge graphs in FL	• Mutual knowledge distillation • Retroactive interference & passive decay for unlearning	• Unlearning for heterogeneous federated KGs • Mutual knowledge distillation reduces bias in local and global models	• High computational cost • Applicability to other graph data needs exploration

Figure 4.25. Visão geral dos métodos de desaprendizagem aproximada em grafos [Xu et al. 2024]

4.7.3. Outras formas de classificação

Outros surveys como [Nguyen et al. 2022] apresentam diferentes critérios de classificação das abordagens de desaprendizado, baseados no nível de agnosticismo e a necessidade de dados de treinamento.

4.7.3.1. Abordagens Model-agnostic

As metodologias de desaprendizagem de máquinas agnósticas em relação aos modelos incluem processos ou frameworks de desaprendizagem que são aplicáveis a diferentes modelos. No entanto, em alguns casos, as garantias teóricas são fornecidas apenas para uma classe de modelos (por exemplo, modelos lineares). No entanto, continuam a ser consideradas agnósticas em relação aos modelos, uma vez que as suas ideias centrais são

aplicáveis a modelos complexos (por exemplo, redes neurais profundas) com resultados práticos.

O trabalho de [Gupta et al. 2021] propuseram um mecanismo de desaprendizado especificamente para pedidos de remoção de dados em streaming. Estes pedidos são também adaptativos, o que significa que os dados a remover dependem do modelo atual de desaprendizado.

Já o trabalho de [Golatkar et al. 2020a] introduz um limite superior computável para algoritmos de aprendizagem baseados no stochastic gradient descent (SGD), especialmente redes neurais profundas. A ideia central baseia-se na noção de perturbação (ruído) para mascarar o pequeno resíduo incorrido pela atualização baseada no gradiente. A ideia é aplicável a outros casos, embora não sejam dadas garantias teóricas.

Ullah et al. [156] continuaram a estudar a desaprendizagem de máquinas no contexto do SGD e dos pedidos de remoção em fluxo contínuo. Provaram que existe um processo de desaprendizagem que satisfaz a desaprendizagem exacta em qualquer momento do pedido de remoção em fluxo contínuo,

4.7.3.2. Abordagens Model-intrinsic

As abordagens intrínsecas ao modelo incluem métodos de desaprendizagem concebidos para um tipo específico de modelos. Embora sejam intrínsecas ao modelo, as suas aplicações não são necessariamente limitadas, uma vez que muitos modelos de aprendizagem automática podem partilhar o mesmo tipo.

O trabalho do [Izzo et al. 2021] propôs um método de desaprendizagem aproximado para modelos lineares e logísticos baseado em funções de influência. Aproximaram o cálculo da matriz Hessiana com uma atualização dos resíduos do projeto, conhecido pela sigla PRU, que combina métodos de gradiente com dados sintéticos. Este método é adequado para esquecer pequenos grupos de pontos de um modelo aprendido. A Figura 4.26 apresenta o pseudocódigo do PRU.

Algorithm 1 The projective residual update

procedure PRU($X, Y, H, \theta^{\text{full}}, k$)
  $\hat{y}'_1, \ldots, \hat{y}'_k \leftarrow \text{LKO}(X, Y, H, k)$
  $S^{-1} \leftarrow \text{PseudoInv}(\sum_{i=1}^{k} x_i x_i^\intercal)$
  $\nabla L \leftarrow \sum_{i=1}^{k} (\theta^{\text{full}\intercal} x_i - \hat{y}'_i) x_i$
  return $\theta^{\text{full}} - \text{FastMult}(S^{-1}, \nabla L)$
end procedure

Figure 4.26. Pseudocódigo do PRU

Motivados pelo regulamento da UE “Direito a ser esquecido”, iniciamos um estudo dos problemas de eliminação de dados estatísticos nos casos em que os dados dos

utilizadores estão acessíveis apenas durante um período de tempo limitado. O artigo de [Li et al. 2020] formularam um caso especial da configuração online em que os dados só estão acessíveis durante um período de tempo limitado, logo não ocorre um processo de treinamento completo. Mais precisamente, o sistema dispõe de uma memória constante para armazenar dados históricos ou um esboço de dados, e tem de fazer previsões num período de tempo limitado. Embora os dados a esquecer possam ser desaprendidos de um modelo em tempo real utilizando um esquema de arrependimento na memória, este processo específico de desaprendizagem só é aplicável à regressão linear.

Já [Schelter et al. 2021] propuseram uma solução de desaprendizagem para árvores extremamente aleatórias, medindo a robustez das decisões de divisão. Uma decisão de divisão é robusta se a remoção de k itens de dados não reverter essa divisão. Note-se que k pode ser limitado, e é muitas vezes pequeno como apenas um em dez mil usuários que querem remover os seus dados de cada vez). Como resultado, o algoritmo de aprendizagem é redesenhado de forma a que a maioria das divisões, especialmente as de alto nível, sejam robustas. Para as divisões não robustas, todas as variantes de sub-árvore são desenvolvidas a partir de todos os candidatos a divisão e mantidas até que um pedido de remoção reveja essa divisão. Quando isso acontece, a divisão é mudada para a sua variante com maior ganho de Gini. Como resultado, o processo de desaprendizagem envolve o recálculo dos ganhos de Gini e a atualização das divisões, se necessário.

4.7.3.3. Abordagens Data-driven

Dentre os trabalhos que propuseram frameworks de desaprendizado cujas decisões são baseadas nos dados. O trabalho de [Huang et al. 2021] apresenta a ideia de ruído de minimização de erros, que leva um modelo a pensar que não há nada a aprender com um determinado conjunto de dados (ou seja, a perda não muda). No entanto, só pode ser utilizado para proteger um determinado item de dados antes de o modelo ser treinado.

Por outro lado, [Tarun et al. 2023] propôs um ruído que maximiza o erro para prejudicar o modelo numa classe de dados alvo (a ser esquecida). No entanto, esta tática não funciona em itens de dados específicos, uma vez que é mais fácil interferir com a previsão de um modelo numa classe inteira do que num item de dados específico dessa classe.

O trabalho de [Zeng et al. 2023] sugeriram um novo método de modelação da influência dos dados, adicionando termos de regularização ao algoritmo de aprendizado. Embora este método seja independente do modelo, requer a intervenção no processo de formação do modelo original. Além disso, só é aplicável a problemas de aprendizagem convexa e a redes neurais profundas.

4.7.3.4. Abordagens Zero-Shot Unlearning

Este cenário ocorre se os dados de treino não forem acessíveis por um método de desaprendizagem, ou seja, a desaprendizagem ocorre com zero amostras de treino. No entanto, não é trivial resolver este problema em casos genéricos.

Esta abordagem foi introduzida no trabalho de [Chundawat et al. 2023b] estudou um cenário deste tipo para a classificação com classes a serem esquecidas, com a ideia de que os resultados do modelo desaprendido devem assemelhar-se aos de um modelo treinado novamente. Além disso, os autores propões duas novas soluções para a desaprendizagem de máquinas com zero disparos, com base em (a) minimização de erros - maximização de ruído e (b) transferência de conhecimentos. Estes métodos removem a informação dos dados esquecidos do modelo, mantendo a eficácia do modelo nos dados retidos. Dentre os benefícios relacionados a privacidade, os autores apontam que esta abordagem oferece uma boa proteção contra os ataques de inversão do modelo e os ataques de inferência de membros.

4.7.3.5. Abordagens One-Shot Unlearning

Para este cenário, o trabalho [Cong and Mahdavi 2022] propôs uma solução de desaprendizagem one-shot, caracterizada por requerer apenas o acesso aos dados a serem esquecidos. A ideia é inspirada na arquitetura de uma rede neural em grafos linear (GNN), em que as não-linearidades numa GNN típica são substituídas por uma matriz de peso único entre camadas convolucionais consecutivas. Apesar da sua extensão linear sobre todas as características dos nós de entrada, essas GNN lineares têm mostrado um desempenho competente, por exemplo, SGC [Wu et al. 2019] e APPNP [Klicpera et al. 2019]. Ao utilizar esta propriedade, os autores propuseram um processo exato de desaprendizagem a nível algorítmico baseado em operações lineares como a projeção e a recombinação.

4.7.3.6. Abordagens Few-Shot Unlearning

A desaprendizagem few-shot consiste em um algoritmo de desaprendizagem que apenas recebe uma pequena parte dos dados a serem esquecidos D_f. Esta definição é útil para os casos em que o conjunto a esquecer contém dados mal rotulados ou se pretende remover os efeitos maliciosos de alguns dados num modelo.

Para este cenário, o trabalho de [Yoon et al. 2022] formula o problema da desaprendizagem com poucas amostras dos dados alvo, especificando as intenções por detrás do pedido de desaprendizagem (por exemplo, desaprendizagem pura e simples, correção de erros de rotulagem, proteção da privacidade), e concebemos uma estrutura simples que (i) recupera um proxy dos dados de treino através da inversão do modelo, explorando plenamente a informação disponível no contexto da desaprendizagem; (ii) ajusta o proxy de acordo com a intenção de desaprendizagem; e (iii) atualiza o modelo com o proxy ajustado. Além disos, Demonstram que o método utilizado por eles, com apenas um subconjunto de dados-alvo, pode superar os métodos de desaprendizagem mais avançados, mesmo com uma indicação completa dos dados-alvo.

4.8. Aplicações de DM

Dentre as aplicações do cotidiano que o desaprendizado de máquinas é extremamente útil, é possível mencionar as ações de segurança contra os ataques cibernéticos. Em muitas situações, a natureza dos ciberataques inclui não só o roubo de dados mas também a

penetração nos modelos de inteligência artificial do cliente, utilizadas pela maioria dos servidores e sistemas atuais. Nesses casos, é introduzida no sistema uma falha que contém dados contraditórios, com os quais o modelo acaba por aprender [Wazid et al. 2022]. O DM é utilizado quando todos os dados hostis têm de ser apagados e o modelo tem de ser reconstruído [Chamola et al. 2023].

Além dos ciberataques, outra aplicação importante envolve os sistemas de recomendação. O trabalho de [Xu et al. 2023] aponta que quando os usuários querem esquecer as suas ações passadas, como términos de relacionamento como no exemplo da 4.27, eles esperam que as plataformas de recomendação apaguem os dados seletivos ao nível do modelo. Idealmente, tendo em conta o histórico de um determinado usuário, o sistema de recomendação pode ser desfeito ou "esquecido", como se o registo não fizesse parte do treinamento. Logo, é desenvolvido o Unlearn-ALS para pedidos de desaprendizado no Netflix.

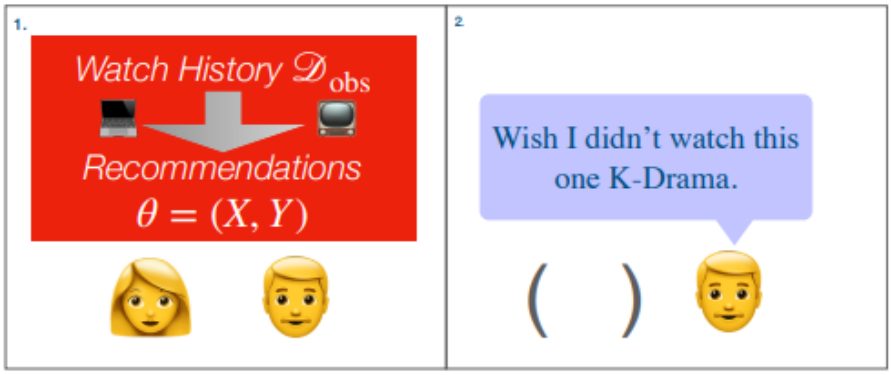


Figure 4.27. Usuário solicita ao Netflix que apague os históricos de dados

Além dos sistemas de recomendação, DM tem tido um papel vital na tradução de idiomas, visto que os algoritmos de desaprendizagem são essenciais para permitir que os modelos invertam palavras e padrões linguísticos previamente adquiridos [Hutchins 1986]. Os métodos DM podem ser utilizados para completar este processo de desaprendizagem. Estes modelos podem fornecer aos clientes traduções mais precisas e atualizadas desaprendizagem de componentes linguísticos desatualizados.

Outra aplicação que vem ganhando destaque na indústria são os modelos de reconhecimento facial baseados em ML, são frequentemente utilizados para controlar o acesso aos espaços de trabalho e impedir a entrada ilegal. As informações de um funcionário têm de ser eliminadas da lista de usuários aprovados após a sua saída da organização. O retreinamento de um modelo com dados novos pode ser computacionalmente exigente. Nesses casos, os dados de um de um empregado ou grupo de empregados podem ser apagados permanentemente utilizando técnicas DM [Rosenfeld 2002].

Portanto, as inúmeras aplicações que podem demandar desaprendizado ratifica a importância deste paradigma para aprimorar modelos de ML, permitindo a remoção seletiva de informações indesejadas. Isso tem implicações importantes em termos de privacidade e segurança.

References

[Bourtoule et al. 2021] Bourtoule, L., Chandrasekaran, V., Choquette-Choo, C. A., Jia, H., Travers, A., Zhang, B., Lie, D., and Papernot, N. (2021). Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 141–159. IEEE.

[Brophy and Lowd 2021] Brophy, J. and Lowd, D. (2021). Machine unlearning for random forests. In *International Conference on Machine Learning*, pages 1092–1104. PMLR.

[Buchele et al. 2016] Buchele, G. T., Teza, P., Müller, I. R. F., and Souza, J. A. d. (2016). Desaprendizagem organizacional: um estudo de campo na universidade federal de santa catarina. *Revista de Administração Contemporânea*, 20:64–83.

[Cao and Yang 2015] Cao, Y. and Yang, J. (2015). Towards making systems forget with machine unlearning. In *2015 IEEE symposium on security and privacy*, pages 463–480. IEEE.

[Capanema 2020] Capanema, W. A. (2020). A responsabilidade civil na lei geral de proteção de dados. *Cadernos Jurídicos, São Paulo, ano*, 21:163–170.

[Chamola et al. 2023] Chamola, V., Goyal, A., Sharma, P., Hassija, V., Binh, H. T. T., and Saxena, V. (2023). Artificial intelligence-assisted blockchain-based framework for smart and secure emr management. *Neural Computing and Applications*, 35(31):22959–22969.

[Chao 2011] Chao, W.-L. (2011). Machine learning tutorial. *Digital Image and Signal Processing*.

[Chen et al. 2021] Chen, M., Zhang, Z., Wang, T., Backes, M., Humbert, M., and Zhang, Y. (2021). When machine unlearning jeopardizes privacy. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pages 896–911.

[Chen et al. 2022] Chen, M., Zhang, Z., Wang, T., Backes, M., Humbert, M., and Zhang, Y. (2022). Graph unlearning. In *Proceedings of the 2022 ACM SIGSAC conference on computer and communications security*, pages 499–513.

[Cheng et al. 2023] Cheng, J., Dasoulas, G., He, H., Agarwal, C., and Zitnik, M. (2023). Gnndelete: A general strategy for unlearning in graph neural networks. *arXiv preprint arXiv:2302.13406*.

[Chundawat et al. 2023a] Chundawat, V. S., Tarun, A. K., Mandal, M., and Kankanhalli, M. (2023a). Can bad teaching induce forgetting? unlearning in deep networks using an incompetent teacher. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 7210–7217.

[Chundawat et al. 2023b] Chundawat, V. S., Tarun, A. K., Mandal, M., and Kankanhalli, M. (2023b). Zero-shot machine unlearning. *IEEE Transactions on Information Forensics and Security*.

[Cong and Mahdavi 2022] Cong, W. and Mahdavi, M. (2022). Grapheditor: An efficient graph representation learning and unlearning approach.

[Dang 2021] Dang, Q.-V. (2021). Right to be forgotten in the age of machine learning. In *Advances in Digital Science: ICADS 2021*, pages 403–411. Springer.

[Du et al. 2019] Du, M., Chen, Z., Liu, C., Oak, R., and Song, D. (2019). Lifelong anomaly detection through unlearning. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1283–1297.

[Ekstrand et al. 2011] Ekstrand, M. D., Ludwig, M., Konstan, J. A., and Riedl, J. T. (2011). Rethinking the recommender research ecosystem: reproducibility, openness, and lenskit. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 133–140.

[Ferreira 1999] Ferreira, A. d. H. (1999). *Novo Aurélio Século XXI: o dicionário da língua portuguesa. rev. e ampl.* Nova Fronteira.

[Gao et al. 2012] Gao, H., Chen, Y., Lee, K., Palsetia, D., and Choudhary, A. N. (2012). Towards online spam filtering in social networks. In *NDSS*, volume 12, pages 1–16.

[Ginart et al. 2019] Ginart, A., Guan, M., Valiant, G., and Zou, J. Y. (2019). Making ai forget you: Data deletion in machine learning. *Advances in neural information processing systems*, 32.

[Golatkar et al. 2020a] Golatkar, A., Achille, A., and Soatto, S. (2020a). Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9304–9312.

[Golatkar et al. 2020b] Golatkar, A., Achille, A., and Soatto, S. (2020b). Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*, pages 383–398. Springer.

[Guo et al. 2019] Guo, C., Goldstein, T., Hannun, A., and Van Der Maaten, L. (2019). Certified data removal from machine learning models. *arXiv preprint arXiv:1911.03030*.

[Guo et al. 2022] Guo, T., Guo, S., Zhang, J., Xu, W., and Wang, J. (2022). Efficient attribute unlearning: Towards selective removal of input attributes from feature representations. *arXiv preprint arXiv:2202.13295*.

[Gupta et al. 2021] Gupta, V., Jung, C., Neel, S., Roth, A., Sharifi-Malvajerdi, S., and Waites, C. (2021). Adaptive machine unlearning. *Advances in Neural Information Processing Systems*, 34:16319–16330.

[He et al. 2021] He, Y., Meng, G., Chen, K., He, J., and Hu, X. (2021). Deepobliviate: a powerful charm for erasing data residual memory in deep neural networks. *arXiv preprint arXiv:2105.06209*.

[Holan and Phillips 2004] Holan, P. M. d. and Phillips, N. (2004). Remembrance of things past? the dynamics of organizational forgetting. *Management science*, 50(11):1603–1613.

[Huang et al. 2021] Huang, H., Ma, X., Erfani, S. M., Bailey, J., and Wang, Y. (2021). Unlearnable examples: Making personal data unexploitable. *arXiv preprint arXiv:2101.04898*.

[Hutchins 1986] Hutchins, W. J. (1986). *Machine translation: past, present, future*. Ellis Horwood Chichester.

[Izzo et al. 2021] Izzo, Z., Smart, M. A., Chaudhuri, K., and Zou, J. (2021). Approximate data deletion from machine learning models. In *International Conference on Artificial Intelligence and Statistics*, pages 2008–2016. PMLR.

[Klicpera et al. 2019] Klicpera, J., Bojchevski, A., and Günnemann, S. (2019). Combining neural networks with personalized pagerank for classification on graphs. In *International conference on learning representations*.

[Koh and Liang 2017] Koh, P. W. and Liang, P. (2017). Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR.

[Koh et al. 2019] Koh, P. W. W., Ang, K.-S., Teo, H., and Liang, P. S. (2019). On the accuracy of influence functions for measuring group effects. *Advances in neural information processing systems*, 32.

[Langley and Carbonell 1984] Langley, P. and Carbonell, J. G. (1984). Approaches to machine learning. *Journal of the American Society for Information Science*, 35(5):306–316.

[Laskov and Šrndić 2011] Laskov, P. and Šrndić, N. (2011). Static detection of malicious javascript-bearing pdf documents. In *Proceedings of the 27th annual computer security applications conference*, pages 373–382.

[Li et al. 2020] Li, Y., Wang, C.-H., and Cheng, G. (2020). Online forgetting process for linear regression models. *arXiv preprint arXiv:2012.01668*.

[Liu et al. 2020] Liu, G., Ma, X., Yang, Y., Wang, C., and Liu, J. (2020). Federated unlearning. *arXiv preprint arXiv:2012.13891*.

[Magdziarczyk 2019] Magdziarczyk, M. (2019). Right to be forgotten in light of regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec. In *6th International Multidisciplinary Scientific Conference on Social Sciences and Art Sgem 2019*, pages 177–184.

[Marchant et al. 2022] Marchant, N. G., Rubinstein, B. I., and Alfeld, S. (2022). Hard to forget: Poisoning attacks on certified machine unlearning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7691–7700.

[Mehrabi et al. 2021] Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., and Galstyan, A. (2021). A survey on bias and fairness in machine learning. *ACM computing surveys (CSUR)*, 54(6):1–35.

[Mehta et al. 2022] Mehta, R., Pal, S., Singh, V., and Ravi, S. N. (2022). Deep unlearning via randomized conditionally independent hessians. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10422–10431.

[Mercuri et al. 2022] Mercuri, S., Khraishi, R., Okhrati, R., Batra, D., Hamill, C., Ghasempour, T., and Nowlan, A. (2022). An introduction to machine unlearning. *arXiv preprint arXiv:2209.00939*.

[Neel et al. 2021] Neel, S., Roth, A., and Sharifi-Malvajerdi, S. (2021). Descent-to-delete: Gradient-based methods for machine unlearning. In *Algorithmic Learning Theory*, pages 931–962. PMLR.

[Nguyen et al. 2020] Nguyen, Q. P., Low, B. K. H., and Jaillet, P. (2020). Variational bayesian unlearning. *Advances in Neural Information Processing Systems*, 33:16025–16036.

[Nguyen et al. 2022] Nguyen, T. T., Huynh, T. T., Nguyen, P. L., Liew, A. W.-C., Yin, H., and Nguyen, Q. V. H. (2022). A survey of machine unlearning. *arXiv preprint arXiv:2209.02299*.

[Pan et al. 2023] Pan, C., Chien, E., and Milenkovic, O. (2023). Unlearning graph classifiers with limited data resources. In *Proceedings of the ACM Web Conference 2023*, pages 716–726.

[Pardau 2018] Pardau, S. L. (2018). The california consumer privacy act: Towards a european-style privacy regime in the united states. *J. Tech. L. & Pol'y*, 23:68.

[Qu et al. 2023] Qu, Y., Yuan, X., Ding, M., Ni, W., Rakotoarivelo, T., and Smith, D. (2023). Learn to unlearn: A survey on machine unlearning. *arXiv preprint arXiv:2305.07512*.

[Ren et al. 2020] Ren, K., Zheng, T., Qin, Z., and Liu, X. (2020). Adversarial attacks and defenses in deep learning. *Engineering*, 6(3):346–360.

[Rosenfeld 2002] Rosenfeld, A. (2002). Face recognition: A literature survey. *UMD, 2002*.

[Schelter et al. 2021] Schelter, S., Grafberger, S., and Dunning, T. (2021). Hedgecut: Maintaining randomised trees for low-latency machine unlearning. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1545–1557.

[Sekhari et al. 2021] Sekhari, A., Acharya, J., Kamath, G., and Suresh, A. T. (2021). Remember what you want to forget: Algorithms for machine unlearning. *Advances in Neural Information Processing Systems*, 34:18075–18086.

[Shibata et al. 2021] Shibata, T., Irie, G., Ikami, D., and Mitsuzumi, Y. (2021). Learning with selective forgetting. In *IJCAI*, volume 3, page 4.

[Sommer et al. 2020] Sommer, D. M., Song, L., Wagh, S., and Mittal, P. (2020). Towards probabilistic verification of machine unlearning. *arXiv preprint arXiv:2003.04247*.

[Sommer et al. 2022] Sommer, D. M., Song, L., Wagh, S., and Mittal, P. (2022). Athena: Probabilistic verification of machine unlearning. *Proceedings on Privacy Enhancing Technologies*.

[Tarun et al. 2023] Tarun, A. K., Chundawat, V. S., Mandal, M., and Kankanhalli, M. (2023). Fast yet effective machine unlearning. *IEEE Transactions on Neural Networks and Learning Systems*.

[Thudi et al. 2022] Thudi, A., Deza, G., Chandrasekaran, V., and Papernot, N. (2022). Unrolling sgd: Understanding factors influencing machine unlearning. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 303–319. IEEE.

[Veale et al. 2018] Veale, M., Binns, R., and Edwards, L. (2018). Algorithms that remember: model inversion attacks and data protection law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2133):20180083.

[Wang et al. 2023] Wang, C.-L., Huai, M., and Wang, D. (2023). Inductive graph unlearning. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3205–3222.

[Warnecke et al. 2021] Warnecke, A., Pirch, L., Wressnegger, C., and Rieck, K. (2021). Machine unlearning of features and labels. *arXiv preprint arXiv:2108.11577*.

[Wazid et al. 2022] Wazid, M., Das, A. K., Chamola, V., and Park, Y. (2022). Uniting cyber security and machine learning: Advantages, challenges and future research. *ICT express*, 8(3):313–321.

[Weber 2011] Weber, R. H. (2011). The right to be forgotten: more than a pandora's box. *J. Intell. Prop. Info. Tech. & Elec. Com. L.*, 2:120.

[Wixted 2021] Wixted, J. T. (2021). The role of retroactive interference and consolidation in everyday forgetting. In *Current issues in memory*, pages 117–143. Routledge.

[Wu et al. 2019] Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019). Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR.

[Wu et al. 2022] Wu, G., Hashemi, M., and Srinivasa, C. (2022). Puma: Performance unchanged model augmentation for training data removal. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 8675–8682.

[Wu et al. 2023] Wu, J., Yang, Y., Qian, Y., Sui, Y., Wang, X., and He, X. (2023). Gif: A general graph unlearning strategy via influence function. In *Proceedings of the ACM Web Conference 2023*, pages 651–661.

[Wu et al. 2020] Wu, Y., Dobriban, E., and Davidson, S. (2020). Deltagrad: Rapid retraining of machine learning models. In *International Conference on Machine Learning*, pages 10355–10366. PMLR.

[Xu et al. 2024] Xu, J., Wu, Z., Wang, C., and Jia, X. (2024). Machine unlearning: Solutions and challenges. *IEEE Transactions on Emerging Topics in Computational Intelligence*.

[Xu et al. 2023] Xu, M., Sun, J., Yang, X., Yao, K., and Wang, C. (2023). Netflix and forget: Efficient and exact machine unlearning from bi-linear recommendations. *arXiv preprint arXiv:2302.06676*.

[Yoon et al. 2022] Yoon, Y., Nam, J., Yun, H., Lee, J., Kim, D., and Ok, J. (2022). Few-shot unlearning by model inversion. *arXiv preprint arXiv:2205.15567*.

[Zeng et al. 2023] Zeng, Y., Wang, J. T., Chen, S., Just, H. A., Jin, R., and Jia, R. (2023). Modelpred: A framework for predicting trained model from training data. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 432–449. IEEE.

[Zhu et al. 2023] Zhu, X., Li, G., and Hu, W. (2023). Heterogeneous federated knowledge graph embedding learning and unlearning. In *Proceedings of the ACM Web Conference 2023*, pages 2444–2454.

Capítulo 5

Testbeds para Pesquisa Experimental em Cibersegurança: Da Teoria à Prática

Michelle Silva Wangham[*‡], Bruno H. Meyer[†], Davi D. Gemmer[*], Khalil G. Q. de Santana[‡], Lucas Rodrigues Frank[§], Luiz Eduardo Folly de Campos[*], Emerson Ribeiro de Mello[¶] e Marcos Felipe Schwarz[*]

Abstract

The current scenario for experimental research in cybersecurity is promising and broad, encompassing various infrastructures and application domains. However, conducting experiments faces significant challenges, such as high costs, operational risks, resource and network management, heterogeneity, capacity and quantity of devices, flexible experiment orchestration, and a large volume of generated experimental data. This chapter aims to present specialized testbeds for conducting cybersecurity experiments, focusing on the MENTORED Testbed, particularly emphasizing the analysis of vulnerabilities, attacks, and defense strategies associated with Internet of Things devices. This testbed is a controlled environment for experimentation and is used as a case study for hands-on demonstrations of theoretical concepts. This chapter describes two DDoS attack experiments, and their results are presented and analyzed.

Resumo

O cenário atual para pesquisa experimental em cibersegurança é promissor e abrangente, englobando diversas infraestrutura e domínios de aplicação. Porém, a condução de experimentos enfrenta uma série de desafios significativos, tais como altos custos, riscos operacionais, gerenciamento de recursos e de redes, heterogeneidade, capacidade

[*]Rede Nacional de Ensino e Pesquisa. Email: michelle.wangham@rnp.br, davi.gemmer@rnp.br, luiz.campos@rnp.br, marcos.schwarz@rnp.br

[†]Universidade Federal do Paraná. Email: bruno.meyer@ufpr.br

[‡]Universidade do Vale do Itajaí. Email: wangham@univali.br, khalil.santana@edu.univali.br

[§]Universidade Federal de Juiz de Fora. Email: lucasrodrigues@ice.ufjf.br

[¶]Instituto Federal de Santa Catarina. Email: mello@ifsc.edu.br

e quantidade de dispositivos, orquestração flexível de experimentos e grande volume de dados experimentais gerado. Este capítulo tem como objetivo apresentar testbeds especializados para a condução de experimentos em cibersegurança, com destaque para o MENTORED Testbed, com um foco particular na análise de vulnerabilidades, ataques e estratégias de defesa associadas aos dispositivos da Internet das Coisas. Este testbed é um ambiente controlado para experimentação e é utilizado como estudo de caso para demonstrações práticas de conceitos teóricos. Dois experimentos com ataques de DDoS são descritos neste capítulo e seus resultados apresentados e analisados.

5.1. Introdução

O relatório de Riscos Globais do *World Economic Forum* 2024 (FORUM, 2024) aponta que os ataques cibernéticos continuam entre os maiores riscos advindos da rápida disseminação da Inteligência Artificial (IA) generativa e de outras novas tecnologias que podem ser facilmente utilizadas em ciberataques, representando uma séria ameaça tanto para as empresas como para a vida pública. Diante da evolução constante das ameaças e dos ataques de segurança, é crucial o desenvolvimento e o aprimoramento contínuo de soluções robustas de cibersegurança.

Desenvolver soluções robustas para prevenir, detectar e mitigar ataques de cibersegurança requer ferramentas e métodos para testá-los e validá-los. Devido aos custos e aos riscos legais, éticos e operacionais, tais como interrupção ou danos às infraestruturas, a realização de experimentações em ambientes reais ou de produção nem sempre são viáveis ou adequadas (SÁEZ-DE-CÁMARA et al., 2023).

De acordo com Goodfellow, Thurlow e Ravi (2018), o cenário atual para experimentação é diverso e abrangente, englobando a proteção de infraestruturas críticas, a segurança de sistemas ciberfísicos, redes celulares, plataformas automotivas, dispositivos IoT e os sistemas de controle industrial. Para pesquisadores e profissionais da área de cibersegurança, a seleção de plataformas adequadas para validar soluções e testar hipóteses de pesquisa é uma tarefa complexa (CHOULIARAS et al., 2021).

Essa busca envolve enfrentar desafios críticos, como a necessidade de encontrar ambientes para experimentação que atendam a requisitos fundamentais, incluindo reprodutibilidade, fidelidade, isolamento, realismo, flexibilidade, escalabilidade e custo (BENZEL, 2011; CHERNYSHEV et al., 2017; PRATES JR et al., 2021).

Na busca por validar sistemas, protocolos e implementações de segurança, os pesquisadores se deparam com a difícil escolha entre simuladores, emuladores e *testbeds*. Embora simuladores e emuladores forneçam ambientes controlados para experimentos reproduzíveis, muitas vezes carecem de realismo. Já os *testbeds* oferecem realismo e escalabilidade, mas podem ser limitados pelo custo e pela disponibilidade de equipamentos reais (GOMEZ et al., 2023). *Testbeds* também enfrentam alguns desafios e nem sempre conseguem reproduzir aspectos de eventos reais, mesmo que tenham a capacidade de reproduzir cenários relevantes. Contudo, a importância dos *testbeds* é refletida pela quantidade abundante de estudos que pesquisam o assunto, além de *datasets* populares obtidos por meio de *testbeds* como o Ton_IoT (MOUSTAFA, 2021), o que será detalhado na Seção 5.2. A impossibilidade de testar soluções em ambientes realistas em larga escala representa uma barreira significativa para empresas, instituições e fornecedores de solu-

ções, resultando em atrasos na implementação de novos recursos e limitando a inovação em cibersegurança.

Este capítulo tem como objetivo apresentar *testbeds* especializados para a condução de experimentos em cibersegurança, com destaque para o MENTORED *Testbed*, desenvolvido no âmbito do MENTORED Project[1], financiado pela FAPESP, e construído sobre o Cluster Nacional da Rede Nacional de Ensino e Pesquisa (RNP). Por fim, com o objetivo demonstrar na prática conceitos teóricos de experimentação em cibersegurança, dois experimentos de ataques de DDoS, utilizando o MENTORED *Testbed*, são descritos e analisados.

O restante deste capítulo está estruturado da seguinte forma. Na Subseção 5.1.1, os desafios para conduzir experimentos em cibersegurança e os requisitos para a concepção de *testbeds* para cibersegurança são analisados. Na Seção 5.2 são apresentados *testbeds* que se destacam na pesquisa experimental em cibersegurança e os compara, considerando suas principais características e limitações. Na Seção 5.3 é descrito o MENTORED *Testbed*, suas principais características e os recursos tecnológicos relevantes que este oferece para os experimentadores. Na Seção 5.4 são apresentados dois casos de uso de ataques DDoS, os roteiros práticos para execução dos experimentos no MENTORED *Testbed*, bem como alguns resultados obtidos. Por fim, na Seção 5.5 são feitas as considerações finais e apresentadas perspectivas futuras.

5.1.1. Desafios para conduzir experimentos em cibersegurança

A realização de experimentos em cibersegurança enfrenta uma série de desafios que dificultam o seu progresso. O gerenciamento de recursos e agendamento em diversas regiões geográficas contribuem para um ambiente complexo, devido à necessidade de manter a comunicação entre os dispositivos de forma sincronizada. Além disso, a instabilidade da rede e o tamanho do sistema podem influenciar na precisão dos resultados, bem como na capacidade de resposta imediata às ameaças (GOMEZ et al., 2023).

Adicionalmente, a capacidade dos dispositivos usados são outro ponto que influencia diretamente na execução dos experimentos que envolvem um número elevado de dispositivos, principalmente por conta dos avanços em Internet das Coisas (IoT) (SÁEZ-DE-CÁMARA et al., 2023). Com capacidades variáveis, os dispositivos com menos recursos ou mais antigos podem não ser capazes de executar tarefas com cargas de trabalho intensas, o que por consequência pode limitar a escala e a complexidade dos experimentos e, desta forma, impossibilitar a simulação de ataques em larga escala de forma adequada (MIRKOVIC; BENZEL, 2012).

Schwab e Kline (2019) destacam outros desafios significativos na área, como a necessidade de sistemas que possam suportar tanto experimentação interativa quanto em lote[2], além de uma orquestração flexível e abrangente para configurar e executar cenários automaticamente. Observa-que, à medida que o tamanho do experimento aumenta, a

[1]https://mentored.dcc.ufmg.br/

[2]Na experimentação interativa, o pesquisador interage diretamente com o sistema ou ferramenta podendo ajustar parâmetros, realizar ações e observar os resultados de imediato. Na experimentação em lote, uma série de testes é automatizada e executada em lote, sem a necessidade de intervenção humana durantes os testes.

complexidade de sua configuração também aumenta, demandando mais tempo para configurar e adquirir recursos. Isso muitas vezes resulta na execução contínua de experimentos para evitar o desperdício de tempo.

Além dos desafios já mencionados, a imensa quantidade de dados gerados durante experimentos em cibersegurança apresenta uma complexidade adicional (SÁEZ-DE-CÁMARA et al., 2023; GOMEZ et al., 2023). Lidar com esse volume de dados requer o emprego técnicas avançadas de análise de dados e inteligência artificial. Essas ferramentas são essenciais para identificar padrões e anomalias, permitindo a detecção e mitigação de ataques em tempo real, enquanto minimiza o impacto nos serviços online.

Para enfrentar esses desafios, diversos pesquisadores e profissionais de cibersegurança estão continuamente desenvolvendo e aprimorando ferramentas e técnicas (GOMEZ et al., 2023). Juntamente a esses avanços, destacam-se o uso de tecnologias de virtualização e computação em nuvem, além do desenvolvimento de algoritmos para análise de grandes conjuntos de dados. Além disso, práticas de automação têm sido implementadas para simplificar o gerenciamento de dispositivos distribuídos. Assim, a colaboração entre instituições acadêmicas e empresas torna-se fundamental para proteger sistemas e dados contra ameaças cibernéticas (MIRKOVIC; BENZEL, 2012).

5.1.2. Requisitos dos *testbeds* de cibersegurança

Segundo Siaterlis, Garcia e Genge (2013), diferentes tecnologias podem ser empregadas para realização de experimentos em cibersegurança, como emuladores, simuladores e ambientes reais. A escolha da tecnologia a ser utilizada depende dos objetivos do experimento, bem como das restrições de recursos, como orçamento, tempo e disponibilidade de equipamentos. Geralmente, os emuladores são mais adequados para experimentos de larga escala, enquanto os simuladores são mais adequados para experimentos de pequena escala. Já os *testbeds* são mais adequados para experimentos que exigem realismo e fidelidade. Em Gomez et al. (2023) é apresentada uma revisão da literatura que descreve a utilização de emuladores, simuladores e ambientes reais (*testbeds*) empregados tanto em pesquisa quanto como ferramenta de ensino na área de redes de computadores.

Os *testbeds* apresentam-se como uma alternativa para a realização de experimentos em cibersegurança, uma vez que seria inadequado realizar experimentos em ambientes de produção ou mesmo replicar cenários reais. Contudo, para que seja possível a execução de experimentos realistas e escaláveis, é necessário que os *testbeds* atendam uma série de requisitos, como indicado por Siaterlis, Genge e Hohenadel (2013), os quais serão detalhados a seguir.

- **Fidelidade** – é desejado que *testbeds* possam replicar com precisão do ambiente real, alvo do estudo, de forma a garantir que os resultados obtidos sejam relevantes e significativos. Nota-se que não é necessário replicar todos os detalhes do ambiente real, mas sim os detalhes essenciais para o estudo em questão. Essa abstração é necessária para garantir que os experimentos sejam executados de forma eficiente, sem comprometer a fidelidade dos resultados;
- **Flexibilidade** – é desejado que o *testbed* possa ser facilmente adaptado para diferentes cenários de experimentos, de forma a permitir que o pesquisador possa

realizar experimentos com diferentes configurações de rede e dispositivos. Assim, o *testbed* deve permitir que o pesquisador defina a topologia da rede, os dispositivos envolvidos, os protocolos de comunicação, entre outros aspectos relevantes para o estudo em questão;

- **Reprodutibilidade** – consiste na capacidade de repetir os experimentos e obter os mesmos resultados ou estatisticamente semelhantes. Para garantir a reprodutibilidade, é necessário que os *testbed* permita ao pesquisador definir os estados inicial e final dos experimentos, bem como os parâmetros de execução dos experimentos que indiquem como os experimentos devem ser executados, saindo de um estado inicial para um estado final;

- **Escalabilidade** – é desejado que o *testbed* possa suportar um grande número de dispositivos, com diferentes capacidades, e que possam estar geograficamente dispersos. A escalabilidade do *testbed* consiste na capacidade de adicionar dispositivos e recursos de forma eficiente, isto é, sem que seja necessário reconfigurar toda a infraestrutura do *testbed*;

- **Isolamento** – é importante que os experimentos sejam executados de forma isolada, de modo que um experimento não afete o outro. O isolamento dos experimentos é fundamental para garantir a segurança e a privacidade dos dados utilizados nos experimentos, bem como para garantir a reprodutibilidade dos resultados;

- **Execução segura** – em experimentos em cibersegurança, entende-se que serão empregadas técnicas e ferramentas de adversários para que o pesquisador possa estudar e entender como os ataques gerados a partir dessas funcionam e como podem ser mitigados. Assim, o *testbed* deve permitir o emprego de tais técnicas e ferramentas, porém sem que isso comprometa a infraestrutura do próprio *testbed* ou de terceiros, como aplicações e serviços externos disponíveis na Internet ou na própria rede local da instituição que mantém o *testbed*.

O MENTORED *Testbed*, apresentado na Seção 5.3, foi projetado para atender a esses requisitos, além de alguns outros relacionados ao acesso em tempo real, a interface com o usuário e gestão de times (PRATES JR et al., 2021). Assim, o MENTORED *Testbed* oferece acesso em tempo real aos dispositivos, para que um usuário possa redefinir, reprogramar e monitorar de forma não intrusiva o tráfego de rede e o estado de cada dispositivo durante a execução dos experimentos. A interface amigável fornecida ao usuário, permite a definição, execução e análise dos experimentos de forma simples e intuitiva. A gestão de times permite a atribuição de permissões e papéis aos usuários por projeto, de forma a garantir a distribuição eficiente das tarefas e a colaboração entre os pesquisadores. O testbed oferece ainda a liberdade para os usuários desenvolverem novas classes de ferramentas que facilitem suas pesquisas experimentais.

5.2. *Testbeds* para experimentação

Nesta seção são apresentados *testbeds* para experimentação que se destacam na literatura ou no seu amplo uso em pesquisas acadêmicas, com intuito de apresentar como estes

podem ser usados em pesquisas aplicadas de cibersegurança e as diversas abordagens que estes oferecem, como virtualização, emulação, conteinerização, dentre outras.

5.2.1. *Cluster* Nacional RNP

O *Cluster* Nacional da RNP é uma infraestrutura de computação em nuvem, implantada sobre recursos do Serviço de *Testbeds*[3] da RNP, distribuídos geograficamente pelo território brasileiro, e oferecida como uma plataforma robusta e flexível em ambiente nativo de nuvem para apoio à experimentação em temas de Tecnologias de Informação e Comunicação (TIC) dentro da pesquisa acadêmica e científica nacional.

Este *cluster* é composto por servidores de computação localizados nos Pontos de Presença (PoPs) da rede *backbone* da RNP, disponíveis em 12 capitais do Brasil. Como pode ser observado na Figura 5.1, a conectividade entre os PoPs é garantida pela infraestrutura de *backbone* da RNP, a Rede IPÊ, proporcionando uma rede estável e de alta velocidade com enlaces de até 100 Gbps para o *Cluster* Nacional com o objetivo de suportar as demandas de rede dos projetos de experimentação. A Figura 5.2 ilustra, por meio de um *heatmap*, as atuais latências (arrendondadas) em milisegundos entre os PoPs da Rede IPÊ.

A pilha de *software* desse ambiente utiliza como sistema operacional a distribuição Ubuntu, por se tratar de uma distribuição Linux bem conhecida e amplamente utilizada no meio acadêmico e científico, bem como por possuir alta estabilidade, suporte de 5 anos e uma boa frequência de atualizações, sendo bi-anual para novas versões LTS da distribuição e semestral para novas versões do *kernel* Linux. Como plataforma de orquestração de contêineres se utiliza o Kubernetes, juntamente com o ecossistema de componentes nativos de nuvem reconhecidos pela CNCF (*Cloud Native Computing Foundation*), e adicionalmente alguns serviços externos como DNS, armazenamento centralizado de dados e repositório privado de imagens de contêineres, possibilitando a orquestração, o compartilhamento e o uso eficiente e escalável dos recursos do *testbed* por diferentes projetos de pesquisa simultaneamente.

Além disso, o *Cluster* Nacional também pode integrar novos componentes, serviços e configurações personalizadas para atender à demandas dos projetos de pesquisa, garantindo um ambiente personalizável, flexível e completo. Como um exemplo dessa flexibilidade, foram adicionados novos componentes de *software* e configurações de *kernel* e sistema operacional para suportar aplicações de Core 5G, se tornando a primeira infraestrutura da RNP com esse suporte, após a implementação e validação de uma arquitetura de rede 5G distribuída geograficamente, com *Core Sites* e *Edge Sites*[4], desenvolvida em um outro cluster similar dentro do Testbeds RNP para atender a um projeto de pesquisa[5]. Um outro exemplo muito interessante é o do projeto KCDN[6], um projeto de pesquisa sobre redes de distribuição de conteúdo (CDNs - *Content Delivery Networks*) usando arquiteturas nativas de nuvem (*cloud native*). O projeto desenvolveu a própria arquitetura de CDN em microserviços considerando a abrangência nacional do *cluster*, e utilizou

[3]https://www.rnp.br/servicos/testbeds

[4]https://github.com/zanattabruno/5G-all-in-one-helm

[5]https://inatel.br/brasil6g/

[6]https://doi.org/10.48448/ejkd-xd29

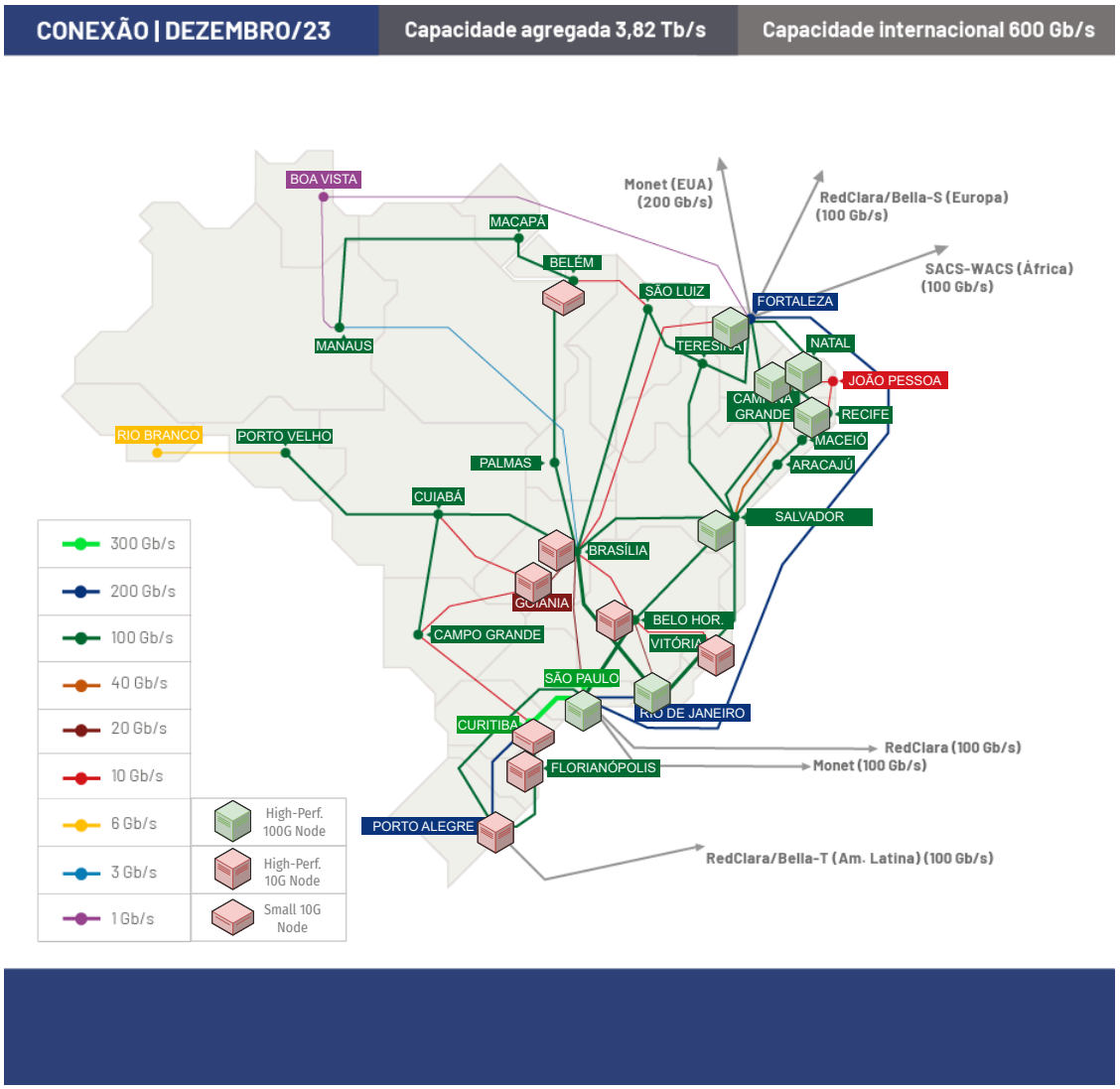


Figura 5.1: Mapa da Infraestrutura do *Cluster* Nacional interconectada pela Rede Ipê.

recursos avançados de DNS e configurações Kubernetes para implantação seletiva por localidade. Foram desenvolvidos 2 serviços DNS externos ao *cluster* para o projeto, sendo 1 deles totalmente configurado pelo KCDN. Algumas funcionalidades desenvolvidas pelo projeto foram posteriormente adicionadas à aplicação eduplay, uma plataforma de *streaming* de video usada pela RNP, e o servidor DNS principal continuou como opção de uso para os demais projetos de pesquisa.

Atualmente, o *cluster* utiliza equipamentos hospedados nos PoPs da RNP, sendo composto principalmente por servidores físicos dedicados, mas também por máquinas virtuais instanciadas no ambiente de TI da RNP em alguns casos. O modelo atual de servidor físico tem como características: 24 núcleos de CPU, 192 GigaBytes de memória RAM e 10 TeraBytes de armazenamento HDD somados a 1 TeraByte de armazenamento NVMe de alto desempenho. Outros modelos de equipamentos podem ser adicionados ao *Cluster* Nacional sob demanda, como por exemplo modelos de servidores menos robustos, com 4 núcleos de CPU, 16 GigaBytes de memória RAM e 300 GigaBytes de armazenamento

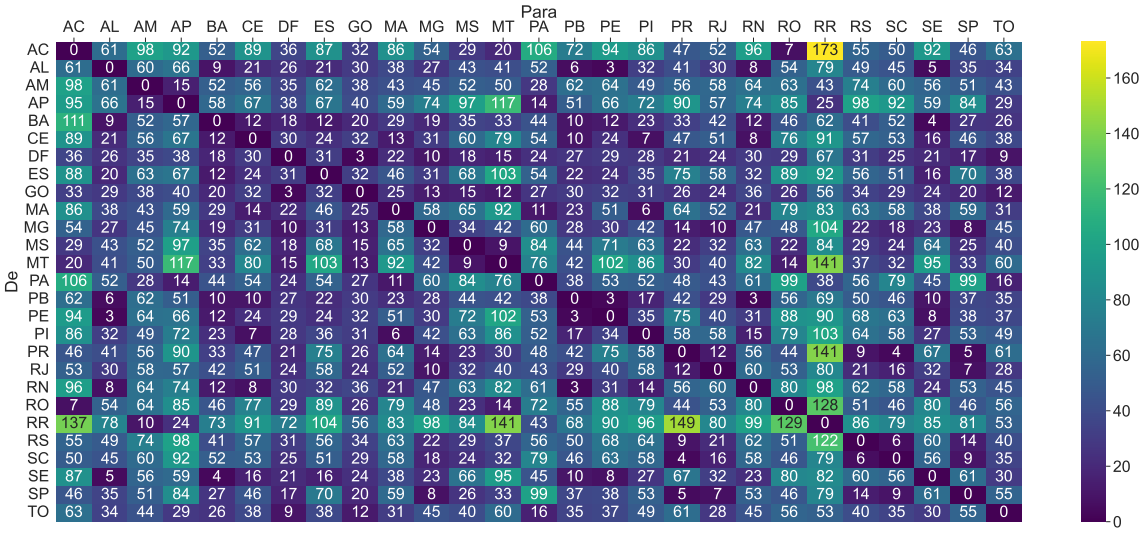


Figura 5.2: Gráfico *heatmap* referente as latências (ms) entre os PoPs da Rede IPÊ.

SSD, ou até mesmo equipamentos especializados como por exemplo dispositivos IoT (Internet das Coisas) de pequeno porte/desempenho e função específica, e *Switches* SDN P4 (*Software Defined Network*). A diversidade de equipamentos do *cluster* garante uma infraestrutura adaptável às diferentes necessidades de pesquisa.

O *Cluster* Nacional é formado por uma quantidade variável de servidores do Serviço de *Testbeds* da RNP, que são alocados como *nodes* do *cluster* conforme as características das demandas de pesquisa corrente, podendo o *cluster* estar com mais ou menos *nodes* ao longo de seu ciclo de vida. Em Julho de 2023, o somatório de recursos alocados para o *cluster* contabilizava um total de mais de 400 núcleos de CPU, mais de 2 TeraBytes de Memória RAM, e mais de 100 TeraBytes de armazenamento. Os equipamentos que são removidos do *cluster* podem ser alocados para projetos que demandem o uso dedicado do servidor, como testes de rede ou de transferências de dados de 100 Gbps, ou para constituírem *clusters* temporários de propósito específico, como por exemplo, para homologação, validação e implantação de novas versões de toda a pilha de software para a evoluçao do próprio Cluster Nacional. O ciclo de vida do *cluster* inclui o *upgrade* da própria versão da pilha Kubernetes, que é executado usando um *cluster* de homologação com a nova versão funcionando simultaneamente ao *cluster* de produção com a versão corrente. Após a validação de funcionamento de todos os projetos de pesquisa na nova versão, é feita a migração das cargas de trabalho, do acesso dos usuários e dos demais *nodes* para o novo *cluster*, fazendo com que o *cluster* de homologação se torne o novo *cluster* de produção, reiniciando todo o ciclo. Os *nodes* em migração para o novo *cluster* também podem ter seus sistemas operacionais atualizados. Com essa estratégia, mesmo as atualizações de toda a pilha do *cluster* ocorrem de forma suave, não incorrendo em qualquer impacto para os usuários.

Os *nodes* do *cluster* e seus respectivos microsserviços em contêineres podem se comunicar utilizando *plugins* de rede (CNI) oferecidos pelo ecossistema Kubernetes. Por padrão é utilizado o CNI Calico, responsável pela conectividade básica do *cluster*. CNIs adicionais como Multus e macvlan são utilizados para permitir a interconexão por meio

de circuitos virtuais criados sobre a Rede Ipê, o *backbone* da RNP, utilizando tecnologias como VXLAN e EVPN, dessa forma permitindo a criação de topologias isoladas definidas pelo usuário e com maior desempenho por utilizarem equipamentos de rede para o encapsulamento e encaminhamento dos pacotes.

A oferta de experimentação no *cluster* é feita através da disponibilização de uma fatia (uma parte) específica dos recursos dessa infraestrutura para cada projeto de pesquisa interessado. A interação com o ambiente é viabilizada através de um nó "*Bastion*", que funciona como ponto de acesso único à *API* de controle do *cluster*, ou até mesmo diretamente a partir de uma máquina remota do usuário.

Cada pesquisador, munido de suas credenciais de acesso, pode então explorar todos os recursos de sua fatia do *cluster* e demais serviços e funcionalidades do ambiente de experimentação de forma isolada dos demais usuários. Cada fatia também pode ser divida em ambientes isolados, através de *namespaces* hierárquicos, a serem utilizados por experimentos distintos, podendo ter seu compartilhamento e permissionamento configurado de forma personalizada. Para garantir uma experiência de uso consistente, são estabelecidas quantidades mínimas de recursos alocados de forma dedicada, garantindo disponibilidade, bem como limites máximos de utilização, evitando a sobrecarga do ambiente. Esta abordagem assegura a segregação dos recursos entre os usuários, ao mesmo tempo em que compartilha esses recursos disponíveis de forma eficiente. Todos esses parâmetros de garantias e limites de recursos podem ser ajustados conforme necessidade e acordo prévio com cada equipe de pesquisa usuária do *testbed*.

Os recursos alocados podem então ser utilizados pelos pesquisadores para a implantação e execução de suas aplicações, seja utilizando arquiteturas tradicionais monolíticas ou adotando abordagens mais modernas como a de microsserviços, aproveitando todas as vantagens de uma infraestrutura distribuída de nuvem. O *Cluster* Nacional também é configurado para aproveitar funcionalidades avançadas da arquitetura Kubernetes, permitindo, por exemplo, que os pesquisadores controlem o comportamento de suas aplicações dentro do *cluster* através da manipulação de alguns parâmetros, e até mesmo as tornem acessíveis externamente ao *cluster*, publicando-as como um serviço.

Para fins de coleta de dados dos experimentos e de uso do *cluster*, o ambiente também proporciona algumas ferramentas de monitoramento que podem auxiliar o experimentaddor, como a ferramenta netdata, instalada em cada *node* para prover métricas gerais de utilização em tempo real, assim como a pilha de monitoramento Kube-Prometheus-Grafana para obtenção de métricas dos objetos internos do *cluster* e suas aplicações. Adicionalmente, os pesquisadores também podem utilizar serviços de suporte externos ao *cluster*, como um armazenamento centralizado, além do armazenamento distribuído do próprio *cluster*, para centralizar coletas de dados ou outras informações em um único lugar facilitando o acesso dos dados para os projeto de pesquisa.

De maneira geral, esse ambiente proporciona o suporte a uma vasta gama de experimentos e projetos de pesquisa. Os temas abordados por grupos de pesquisa no Cluster Nacional, variam desde temas e questões já tradicionais, como Redes, Internet, Redes Programáveis (SDN), Monitoramento e Orquestração de Recursos, até a tópicos mais avançados de pesquisa, como Computação em Nuvem, Computação de Borda, Arquitetura de Microsserviços, Funções Virtualizadas de Rede (NFV), Redes de Distribuição de

Conteúdo (CDN), Segurança Cibernética, IoT (Internet das Coisas), Redes 5G e *Blockchain*. Como resultado, o *Cluster* Nacional tem desempenhado um papel importante no apoio à pesquisa em TIC, contribuindo com muitos projetos de pesquisa desde 2019.

A infraestrutura do *Cluster* Nacional oferece uma série de vantagens notáveis para os pesquisadores. Destacam-se a vasta quantidade de recursos computacionais disponíveis, distribuídos geograficamente por todo o território nacional, abrangendo todas as cinco regiões. Essa distribuição geográfica permite aos pesquisadores explorar essa característica em seus experimentos. Além disso, o ambiente de experimentação é baseado em um sistema tecnológico real, utilizando Kubernetes e o ecossistema CNCF, amplamente reconhecidos e utilizados em ambientes de produção e no mercado em geral. A escalabilidade flexível dos *nodes* e do ambiente como um todo é uma característica fundamental. O *cluster* também oferece aplicações de monitoramento integradas, bem como a modularização de componentes, que proporciona flexibilidade para personalização e integração de novos componentes pelos próprios usuários. Ademais, a interconexão é facilitada através de circuitos virtuais pelo *backbone* da RNP, garantindo taxas de vazão de até 100 Gbps em alguns casos. Essas características combinadas, proporcionam um ambiente adequado para a pesquisa e experimentação em TIC.

5.2.2. DeterLab e MergeTB

O DeterLab (WROCLAWSKI et al., 2016) é um *testbed* inicialmente criado para experimentação na área de cibersegurança, com capacidade de executar experimentos em grande escala, incluindo a execução de ataques e defesas em redes, sistemas distribuídos, sistemas cibernéticos-físicos e sistemas de controle industrial. Iniciamos esta seção descrevendo o DeterLab em seu estado antigo, e, em seguida, apresentamos a nova plataforma adotada pelo projeto, denominada MergeTB.

Concebido em 2004 com base no código de gerenciamento do Emulab[7] da Universidade de Utah, o DeterLab é um dos *testbeds* mais renomados da literatura, financiado pelo Departamento de Segurança Interna (*US Department of Homeland Security*), Fundação Nacional de Ciência (NSF, *National Science Foundation*) e a Agência de Pesquisa de Projetos Avançados de Defesa (DARPA, *Defense Advanced Research Projects Agency*).

A arquitetura do DeterLab é baseada em um modelo de experimentação federada, de forma que é possível interligar outros *testbeds*. Além disso um grupo de políticas permite que cada provedor de infraestrutura controle sobre como seus recursos podem ser utilizados. Assim é possível a execução de experimentos federados, utilizando recursos de múltiplas fontes para possibilitar um ambiente de experimentação coesivo, os quais são descritos em um formato baseado no *Network Simulator* (NS) utilizado pelo Emulab.

A infraestrutura do DeterLab é composta por um conjunto de servidores x86-64, interconectados por múltiplas interfaces Ethernet em *switches* com suporte a VLANs. Cada nó é capaz de executar múltiplos experimentos simultaneamente, sendo que cada experimento é isolado dos demais e da Internet. Além disso, cada nó permite ao experimentador escolher um dos cinco níveis de fidelidade para a execução do experimento, variando desde a alocação de um nó físico por completo para um experimento, passando

[7]https://www.emulab.net

por nós representados por máquinas virtuais QEMU, contêineres OpenVZ, processos virtualizados (ViewOS), e até a execução de *threads* (*co-routines*) em um nó físico.

A flexibilidade para escolha do nível de fidelidade permite ao experimentador escolher o nível mais adequado para o experimento em questão, sendo que os níveis mais altos de fidelidade impactam diretamente na escalabilidade do experimento. Contudo, com um nível de fidelidade mais baixo, é possível estudar fenômenos complexos envolvendo 100.000 ou mais nós, considerando que o experimentador atenha-se aos aspectos mais interessantes ao experimento em questão.

O *testbed* foi amplamente utilizado por pesquisadores, com mais de 2.600 usuários de pesquisa e educação ao redor do mundo, além de 47 instituições em 6 países (MIRKOVIC; BENZEL, 2012). Como ferramenta de ensino, o ambiente de experimentação provido pelo *testbed* pode ser utilizado para a análise de ameaças de segurança e suas defesas, com total liberdade de modificar o sistema operacional utilizado e as ferramentas aplicadas. Existe também a possibilidade de execução de uma série de exercícios pré-definidos pelo docente ou pelo próprio DeterLab, como por exemplo, uma introdução ao Linux e ao *testbed*, ataques de injeção SQL, permissionamento e *firewalls*, computação forense, detecção de intrusão em redes, ataques *Man-in-The-Middle* (MiTM), worms, botnets, e ataques DDoS diversos. Por fim, resaltamos que o *testbed* continua em funcionamento para finalidades educacionais.

Atualmente o DeterLab está em um processo de modernização, começando pela adoção de uma nova plataforma chamada MergeTB[8] e migrando para a infraestrutura do projeto SPHERE[9], que também é financiado pela NSF. Apesar desta migração, o conteúdo didático para execução de experimentos educacionais ainda está acessível por meio do antigo portal do DeterLab[10], e este continua sendo o portal para tais tipos de usuários. No entanto, o site do projeto recomenda que usuários utilizem o novo portal[11] para fins de pesquisa experimental.

A arquitetura do MergeTB é descrita em Goodfellow, Thurlow e Ravi (2018), a qual descreve o *testbed* como sendo composto por diversos serviços *stateless*. Tais serviços ao receber requisições ou dados por meio de uma interface *protobuf*, realizam operações como a realização (alocação) e materialização (instanciação) dos recursos solicitados pelo pesquisador. Durante o processo de materialização, diversos serviços denominados *drivers* são invocados para configurar recursos específicos, reservando eles temporariamente para a execução das tarefas necessárias, e, desta maneira, evitando problemas relacionados ao acesso simultâneo por múltiplos *drivers*. Além deste artigo, a arquitetura do MergeTB também é descrita em sua documentação disponível *online*[12] e em seu *blog*[13], a qual detalha aspectos adicionais como *softwares* específicos utilizados na criação do *testbed*.

[8]https://mergetb.org
[9]https://sphere-project.net
[10]https://www.isi.deterlab.net/newproject.php
[11]https://launch.mod.deterlab.net/
[12]https://mergetb.org/docs/
[13]https://mergetb.org/blog/2021/05/01/merge-1.0/

5.2.3. FIT IoT LAB

O FIT IoT-LAB (ADJIH et al., 2015) é um *testbed* de acesso aberto projetado para experimentação em larga escala, focado em dispositivos IoT sem fio. Com mais de 2.700 dispositivos IoT e 117 robôs móveis, distribuídos em seis locais na França, o objetivo do *testbed* é servir como ferramenta científica para o estudo e avanço de tecnologias sem fio, bem como de tópicos relacionados, como mobilidade, por exemplo.

O *testbed* conta com diferentes tecnologias de comunicação, incluindo 802.15.4, BLE, LoRa, rádios operando em frequências Sub-GHz e *Ultra Wide Band* (UWB). Os dispositivos contam com diferentes sistemas operacionais e são reconfiguráveis e reprogramáveis, permitindo que os usuários substituam o *firmware* conforme necessário. No momento, o *testbed* abrange 18 plataformas de IoT.[14].

O processo para a execução de experimentos envolve inicialmente a reserva de um conjunto de nós, seguido pela configuração desses nós, que pode incluir a carga de um *firmware* específico. Uma vez configurados, o experimento é então executado. Os experimentadores têm a opção de utilizar uma *API* para controlar os dispositivos ou acessar uma aplicação *web* para visualização dos experimentos em andamento.

O *hardware* dos dispositivos IoT consiste em uma placa de experimentação conhecida como *Open Node* (ON), que é reprogramável pelo usuário, e está conectada a um mini computador denominado *Gateway* (GW). O *Gateway* atua como intermediário na comunicação entre o *Open Node* e um *hardware* de controle autônomo chamado de *Control Node* (CN). O *Control Node* é responsável por coordenar a reprogramação, inicialização, *resets*, análise de consumo de energia, manipulação dos sensores, inspeção do tráfego e injeção de pacotes na rede do *Open Node*. A utilização de um microcontrolador dedicado no CN proporciona a vantagem de garantir a execução em tempo real das atividades de medição nos experimentos (ADJIH et al., 2015).

Para plataformas IoT baseadas em microcontrolador, o *Control Node* (CN) oferece ao experimentador acesso em tempo real aos dados dos sensores, bem como a capacidade de realizar ações como *sniffing* e injeção de pacotes na rede do dispositivo. Em plataformas que operam com o sistema operacional Linux, além das facilidades oferecidas para dispositivos baseados em microcontrolador, os experimentadores têm acesso a um sistema de arquivos montado no espaço de usuário e disponível via NFS (*Network File System*), permitindo a persistência de dados entre execuções de experimentos.

O FIT IoT-Lab emergiu como um *testbed* amplamente reconhecido e utilizado por pesquisadores, tendo sido citado em mais de 500 artigos científicos desde sua publicação em 2015. Embora seu foco principal não seja cibersegurança, o *testbed* tem sido empregado em diversos casos de uso. Há uma percepção de que o FIT IoT-Lab oferece potencial para experimentos relacionados à cibersegurança em redes IoT, incluindo ataques e defesas.

5.2.4. Gotham Testbed

Construído sobre o emulador de redes GNS3, o *testbed* Gotham (SÁEZ-DE-CÁMARA et al., 2023) é um *middleware* e conjunto de *scripts* projetados para gerar cenários e realizar

[14]https://www.iot-lab.info/docs/boards/overview/

ataques usando o *testbed*. Este *testbed* tem como escopo a experimentação em dispositivos IoT, bem como a geração de *datasets* contendo capturas de pacote e registros (*logs*) de aplicações. Uma contribuição notável deste trabalho é sua ênfase na reprodutibilidade. Ao contrário de muitos *testbeds* fechados e difícil acesso, o Gotham disponibiliza seu código fonte[15], possibilitando sua reprodução. Embora diversos conjuntos de dados provenientes de *testbeds* estejam amplamente disponíveis na literatura, os próprios *testbeds* raramente são publicados. Essa transparência e disponibilidade do código-fonte são essenciais para promover a reprodutibilidade e a colaboração na pesquisa em IoT e segurança cibernética.

A arquitetura deste *testbed* pode ser descrita como operando em três camadas. A camada mais inferior trata dos executores, isto é, soluções como Docker, QEMU, Dynamips, responsáveis por executar aplicações e imagens de dispositivos, tais como VPCS (*Virtual PC Simulators*), roteadores, *switches*, dentre outros. Essa camada é controlada pela segunda camada, que utiliza o GNS3, um software de código aberto capaz de interconectar dispositivos em uma topologia virtual. Essa topologia pode ser visualizada e controlada por meio de sua interface gráfica ou da *API* REST do próprio GNS3. A terceira camada do *testbed* é composta por diversos subcomponentes.

O cenário incorporado no *testbed* conta com 140 dispositivos, distribuídos em três camadas: *edge*, *network*, e *cloud*. Na primeira camada, encontram-se os dispositivos IoT e IIoT[16], distribuídos em duas zonas: zona de ameaças e zona da cidade. Dentro da zona de ameaças, o primeiro grupo é subdividido em três agentes de ameaças (*threat actors)*: *Maroni*, *Falcone* e *Calabrese*, sendo cada um representado por um modelo de ameaça distinto. Na zona da cidade, também há uma subdivisão em quatro subgrupos, cada um representando locais distintos: o Museu de História Natural (contendo sensores de monitoramento e câmeras IP); o bairro *Bristol* (contendo casas com mecanismos automação residencial e câmeras IP); a fábrica de aço *Rennington* (a qual envia telemetria de equipamentos industriais para a zona *cloud*); e, por último, a central de energia de Gotham (uma rede industrial que também envia telemetria para monitoramento de sistemas hidráulicos e de geração de energia).

A segunda camada do cenário é a zona de redes (*network*), que interconecta os diversos dispositivos mencionados anteriormente por meio de 30 switches e 10 roteadores, formando um *backbone* que orquestra rotas através do protocolo OSPF. Por fim, a zona de nuvem (*cloud*) é responsável por fornecer serviços para as camadas anteriores, como DNS, NTP, dentre outros.

No que diz respeito à experimentação de segurança, os agentes de ameaça realizam uma variedade de ataques na infraestrutura modelada. Diversos tipos de ataques e comportamentos são modelados, incluindo o uso da *botnet* Mirai[17] e a central de comando e controle (C&C) Merlin[18]. Por meio destas ferramentas, são executados ações como *scans* de rede, ataques de força bruta (usando uma lista de credenciais pré-definida), transferência arquivos maliciosos, coleta de dados, ataques DDoS (10 tipos, incluindo ataques de *flood* UDP, DNS, TCP SYN, TCP ACK, TCP *stomp*, GRE e HTTP), execução de

[15]Código fonte Gotham *testbed*: https://github.com/xsaga/gotham-iot-testbed
[16]IIoT: Industrial Internet of Things
[17]Mirai *botnet*: https://www.cloudflare.com/pt-br/learning/ddos/glossary/mirai-botnet/
[18]Merlin C&C: https://github.com/Ne0nd0g/merlin

código remoto (RCE), assim como ataques a fraquezas em protocolos específicos, como MQTT e CoAP, usando ferramentas como MQTTSA e SlowTT-Attack. Algumas alterações no código fonte Mirai foram necessárias para sua execução no ambiente isolado do *testbed*, como troca do endereço DNS *hardcoded* (8.8.8.8) para o endereço DNS do servidor do *testbed*, dentre outros.

Sáez-de-Cámara et al. (2023) faz uma avaliação dos requisitos do *testbed*. Em relação a reprodutibilidade, destaca-se que o uso de Dockerfiles[19] contribui para atingir este requisito, uma vez que estes arquivos descrevem todas as dependências e configurações dos dispositivos. Além disto, também é examinada a emulação de enlaces, uma vez que dispositivos IoT se comunicam por meio de diversos tipos de enlaces. Nesse sentido, o *testbed* deve ser capaz de representar enlaces com qualidade variável, incluindo latência e vazão. Esse requisito é atendido por meio da utilização da ferramenta `tc` do Linux.

Com relação a emulação de recursos de hardware, demonstrou-se que limites de recursos impostos via Docker resultam em métricas consistentes na ferramenta *stress-ng*. Sobre o requisito de escalabilidade, os autores indicam que o consumo de memória RAM é linear de acordo com o número de dispositivos emulados via Docker e QEMU. Por fim, são descritos os critérios de fidelidade e heterogeneidade em relação à execução de cenários com e sem atacantes. Isso inclui uma topologia complexa, diversidade de protocolos e configurações de aplicações, além da capacidade de executar ataques e monitorar o tráfego de rede e registros (*logs*) de aplicações.

5.2.5. GENI e FABRIC

GENI *testbed* (BERMAN et al., 2014) tem como principal característica sua arquitetura distribuída e federada em uma escala nacional nos Estados Unidos. Operando por mais de uma década, o projeto foi descontinuado em 2023, sendo sucedido pelo FABRIC[20]. Apesar do encerramento, é importante entender a arquitetura do GENI, dada sua influência significativa na literatura. Posteriormente, o seu sucessor, o FABRIC, é apresentado.

A arquitetura do GENI baseava-se fortemente na virtualização, o que permitia o compartilhamento eficiente de recursos, tanto computacionais quanto de armazenamento. Sua rede também dispunha de recursos virtualizados e programáveis por meio de uma rede definida por software (SDN, *Software Defined Network*) ou VLANs, além de oferecer conectividade WiMAX em determinados pontos de presença.

A virtualização no *testbed* GENI era implementada por meio de diversas tecnologias, como Linux Vserver[21], KVM (Kernel-based Virtual Machine) ou OpenVZ. O *testbed* também contava com dispositivos físicos, sem virtualização. Além disto, a SDN do *testbed* era composta por componentes utilizando o padrão OpenFlow. A federação GENI ocorria por meio de diversas camadas de abstração, autenticação e autorização. Uma das principais camadas de abstração são as 'agregações' do *testbed*, que representavam os gestores de um grupo de recursos dentro do *testbed*.

[19]Dockerfiles: são arquivos de instruções para a construção de uma imagem de contêiner por meio da ferramenta Docker

[20]https://learn.fabric-testbed.net/knowledge-base/transitioning-from-geni-to-fabric/

[21]http://linux-vserver.org/Welcome_to_Linux-VServer.org

O *testbed* também disponibilizava uma *API* aberta para permitir o desenvolvimento de múltiplas soluções de interação com o ambiente. Tais ferramentas ofereciam funcionalidades, como descoberta de recursos do ambiente, reserva de tais recursos, dentre outros. Um exemplo disso era o ambiente web chamado *Flack*, que permitia a descoberta e configuração de recursos, bem como a criação de topologias por meio da funcionalidade de "clicar e arrastar". Outra ferramenta, denominada *Omni*, servia como uma interface de linha de comando (CLI) de baixo nível para a execução automatizada de tarefas. Já *Myslice* era uma ferramenta Web que auxiliava na identificação dos recursos computacionais mais adequados para um determinado experimento. O *Gush* era utilizado como ambiente de gestão de execução de experimentos, enquanto o *Stork* permitia a implantação e configuração de *softwares* dos experimentos em larga escala. Por fim, o *framework OMF* oferecia o controle, gestão e medição dos experimentos.

FABRIC (BALDIN et al., 2019) é um *testbed* de escala nacional nos Estados Unidos e sucessor do GENI. O principal objetivo deste *testbed* é fornecer uma infraestrutura de experimentação programável, não somente nos dispositivos *edge*, mas também na própria rede que interconecta esses dispositivos. Essa programabilidade é alcançada, em parte, pela utilização de ativos de rede com suporte à tecnologia P4 [22], que é uma linguagem de domínio específico (DSL) para a definição de como um caminho de dados deve processar determinado pacote.

Outro aspecto importante da infraestrutura FABRIC é sua conectividade. Em 2023, o *testbed* inaugurou o TeraCore[23], uma rede com uma topologia de anel capaz de atingir velocidades de 1.2 Tbps por meio de conexões de fibra óptica. Além disto, o *testbed* conta com outros recursos, como FPGAs, GPUs, centenas de núcleos e terabytes de memória RAM, possibilitando experimentação em larga escala.

Cunha Pontes et al. (2023) realizam uma análise deste *testbed* sob a perspectiva do experimentador. Eles descrevem o fluxo de experimentação como: (*i*) criação de um *slice*, (*ii*) automação da configuração dos nós do experimento como roteadores e nós *edge*, e (*iii*) execução do experimento. No FABRIC, um *slice* representa a alocação de um determinado tipo de recurso do *testbed*, como uma máquina virtual, com recursos específicos e vinculada a um projeto no *testbed*. Para a configuração dos dispositivos, os autores recomendam a utilização de ferramentas de automação, como Ansible, que permitem criar um ambiente de experimentação reproduzível. Além disso, os autores destacam que há total liberdade para a modificação do sistema operacional em execução, inclusive permitindo a execução de contêineres dentro dos nós, se desejado.

5.2.6. Fed4Fire+

O projeto europeu Fed4Fire+, concluído em 2021, representou uma significativa iniciativa de pesquisa e desenvolvimento. Seu principal objetivo foi aprimorar a federação de *testbeds* já estabelecida do Fed4FIRE[24], fornecendo uma infraestrutura facilitadora para ex-

[22]https://p4.org/

[23]https://learn.fabric-testbed.net/knowledge-base/nsf-fabric-project-announces-groundbreaking-high-speed-network-infrastructure-expansion/

[24]O projeto inicial Federation for Future Internet Research and Experimentation (Fed4FIRE), financiado pela União Europeia e executado de 2012 a 2016, foi um projeto integrador que abordou o tema de pesquisa e experimentação na Internet do Futuro.

perimentação em áreas-chave, como SDN, IoT e computação em nuvem[25](DEMEESTER et al., 2022). O principal objetivo da federação é proporcionar aos pesquisadores, desenvolvedores e empresas a oportunidade de testar e validar novas tecnologias e serviços em um ambiente realista e controlado.

O portal Fed4FIRE permanece acessível[26] e é mantido pela IMEC, uma instituição de pesquisa belga. Ele oferece acesso a uma variedade de recursos na federação de *testbeds*, incluindo computação, armazenamento e redes, juntamente com ferramentas de gestão e monitoramento. Isso permite que os usuários conduzam testes em larga escala, reproduzindo cenários complexos e avaliando o desempenho, segurança e confiabilidade de suas soluções.

O portal Fed4FIRE+ apresenta e oferece uma gama de tecnologias (cabeadas, sem fio, redes móveis, IoT, SDN, Cloud, Big Data, IA, entre outras) e os dezoito *testbed* federados, incluindo descrições, localizações, configurações e documentações[27]. Além disso, uma funcionalidade útil é o monitoramento, que permite visualizar quais *testbeds* estão conectados no momento[28]. A documentação necessária para criação de conta de acesso, configuração, execução e análise dos resultados dos experimentos permanece sendo atualizada pela comunidade da federação[29]. Por fim, o portal fornece um guia sobre como integrar novos *testbeds* à federação Fed4FIRE+.

5.2.7. *Testbeds* para geração de *datasets*

Esta seção tem como propósito apresentar alguns *testbeds* dedicados à geração de *datasets* para pesquisa. Esses ambientes de experimentação são projetados para coletar dados, como tráfego de rede, telemetria de sensores e registros de aplicação (*logs*), por meio de cenários controlados. Os *datasets* gerados por esses *testbeds* têm uma ampla gama de usos, sendo frequentemente utilizados na concepção e avaliação de ferramentas de detecção e prevenção de intrusões (IDS/IPS), entre outros propósitos científicos.

TON_IoT

TON_IoT (MOUSTAFA, 2021) é um *testbed* dedicado e focado na produção de um *dataset* para pesquisa. O *testbed* é organizado em três camadas: camada *edge* – onde são dispostos dispositivos físicos como sensores, *smartphones*, *smart TVs*, além de outros dispositivos como *workstations* e *laptops*; camada *fog* – utiliza um *hypervisor* para a gestão de recursos virtualizados, por meio da plataforma NSX-VMware, que possui recursos de rede definida por software (SDN), virtualização de funções de rede (NFV), e orquestração de serviços; camada *cloud* – contêm um *dashboard* MQTT para coleta de dados de telemetria IoT, um *website* PHP vulnerável para servir como alvo de ataques de injeção e outros serviços nas *clouds* Microsoft Azure e *Amazon Web Services* (AWS).

O *dataset* TON_IoT compreende dados de telemetria de sensores, assim como

[25]https://www.tu.berlin/av/forschung/projekte/fed4fireplus
[26]https://portal.fed4fire.eu/
[27]https://portal.fed4fire.eu/explore/discover
[28]https://fedmon.fed4fire.eu/overview/
[29]https://doc.fed4fire.eu/#

capturas de pacote dos sistemas implantados. Entre os eventos contidos no *dataset*, encontram-se ataques como *scanners* (Nmap, Nessus), negação de serviço (DoS e DDoS, usando scripts em *Python*), *ransomware*, *backdoor*, injeções SQL (Sqlmap), *cross-site scripting*, *cracking* de senhas e ataques Man-in-The-Middle (Ettercap). Este *dataset* é então processado por meio de ferramentas como Zeek[30], extraindo 44 atributos dos fluxos de dados do *dataset*. A partir destes dados, quatro modelos de inteligência artificial são confeccionados: *Gradient Boosting Machine* (GBM), *Random Forest*, *Naive Bayes*, e *Deep Neural Network*, atingindo um *F1-score* de 0.99, no caso do *Random Forest*.

Em Alsaedi et al. (2020), são detalhadas outras características do *testbed*, como por exemplo, a descrição de *scripts* Javascript que simulam o comportamento de dispositivos como uma geladeira, termostato, e GPS, interligados por meio da ferramenta Node-RED. As informações geradas pelos sensores são enviadas por meio do protocolo MQTT para um *broker* Hive-MQTT na camada de nuvem. Além disto, o *pipeline* de processamento dos dados é apresentado. Os dados gerados pelo *tested* são catalogados (*labelling*), é feita a divisão 80/20% entre treinamento e avaliação, ocorre a estratificação dos dados usando a técnica *k-fold* e o processamento por meio de outros algorítimos de inteligência artificial, tais como *Logistical Regression*, *Linear Discriminant Analysis*, *k-Nearest Neighbour* (kNN), *Classification and Regression Trees* (CART), *Random Forest*, *Naïve Bayes*, *Support Vector Machine*, *Long Short-Term Memory* (LSTM). Os resultados obtidos são variados, conforme o tipo de sensor incluso no *dataset* em questão.

Bot_IoT

Em Koroniotis, Moustafa, Sitnikova et al. (2019), é apresentado o Bot-IoT, um *dataset* construído a partir de um *testbed* dedicado, que utiliza máquinas virtuais utilizando o VMWare EXSi. O cenário utilizado conta com 4 máquinas virtuais utilizando do sistema operacional Kali (executando *scans* de rede, ataques DDoS e outros ataques típicos de *botnets*). Uma VM que utiliza o Windows 7, além de uma instância do Metasploitable[31]. O cenário também conta com uma VM com servidor Ubuntu, configurada com diversos serviços rede, como DNS, e-mail, FTP, HTTP, e SSH, além de dispositivos IoT simulados por meio da utilização de *scripts* e da ferramenta Node-RED. Além do servidor, há um dispositivo Ubuntu *mobile*, um servidor para o monitoramento do tráfego de rede, além de outros ativos de rede como um roteador pfSense e um firewall.

O *dataset* contém diversos tipos de fluxos de dados, abrangendo uma variedade de ataques e sensores que utilizam protocolos como o MQTT. Esses dados são coletados no formato *pcap*, e os fluxos de dados contidos nessa captura são analisados usando a ferramenta Argus. Em seguida, esses dados são persistidos em um banco de dados MySQL para etapas adicionais de processamento. É importante destacar que a etiquetagem (*labelling*) dos fluxos como maliciosos ou não é realizada com base nos endereços IP, ou seja, os fluxos provenientes dos IPs atribuídos às máquinas atacantes são marcados como tráfego malicioso.

[30]Zeek: https://zeek.org/

[31]Metasploitable é uma máquina virtual que intencionalmente contem um grande número de vulnerabilidades, servindo então como um alvo para ataques diversos.

Em (KORONIOTIS; MOUSTAFA; SITNIKOVA et al., 2019) são apresentadas as etapas para a criação de modelos de aprendizado de máquina para a classificação dos fluxos. Diante da grande quantidade de dados (69GB), foram selecionados os 10 melhores atributos do *dataset* por meio de métodos estatísticos, como análise de entropia e correlação de atributos. Estes atributos foram utilizados em modelos como SVM (*Support Vector Machine*), RNN (*Recurring Neural Network*) e LSTM (*Long Short-Term Memory*). Constata-se que há uma boa acurácia dos modelos empregados para a detecção da existência de ataques (classificação binária) e classificação do tipo de ataque. Contudo, o ataque de exfiltração de dados possuiu o pior índice de classificação (multi-classe).

5.2.8. Considerações sobre os *testbeds*

Tendo em vista os *testbeds* apresentados anteriormente, esta seção visa realizar uma comparação entre eles, destacando seus pontos fortes e, ao mesmo tempo, suas limitações de escopo ou arquitetura.

Primeiramente, em relação aos *testbeds* apresentados, observa-se uma ampla variedade de escalas. Projetos como Gotham possuem uma arquitetura geograficamente centralizada, onde os diversos nós modelados nos experimentos estão nos mesmos dispositivos físicos ou localizados no mesmo ambiente. Por outro lado, outros *testbeds* na literatura adotam uma arquitetura distribuída (DeterLab, GENI e Fed4FIRE+), e alguns incluem recursos como federação, permitindo a interconexão de recursos de diferentes entidades (Fed4FIRE+ e FABRIC).

A fidelidade é outro aspecto crucial para avaliar um *testbed*. Neste contexto, destaca-se a abordagem do projeto DeterLab, que oferece múltiplos níveis de representação de nós. Isso permite aos pesquisadores fazer compensações entre a fidelidade de cada componente no experimento e a escalabilidade. Por outro lado, outros *testbeds*, como o apresentado no FIT IoT, optam por utilizar apenas dispositivos físicos para representar os nós do experimento. Embora esta abordagem proporcione maior fidelidade, também apresenta desafios, como a manutenção, escalabilidade limitada e a necessidade de substituir dispositivos para acompanhar novos paradigmas e tecnologias em estudo.

Já em relação a flexibilidade, os *testbeds* apresentados permitem a configuração de diversos tipos de experimentos. Tal flexibilidade é alcançada por meio da utilização de redes e dispositivos programáveis, como por exemplo no caso do GENI, DeterLab, FABRIC, Fed4FIRE. Frisamos outro aspecto importante para a flexibilidade, a programabilidade da rede interconectado os nós de um experimento. Tal programabilidade pode ser alcançada de diversas maneiras, contudo destacamos técnicas de redes definidas por software (SDN) empregadas por protocolos como OpenFlow. Este protocolo permite ter uma malha de rede configurável e inteligente, permitindo a criação de cenários com topologias dinâmicas.

Em relação ao isolamento, observa-se uma variedade de abordagens, com a virtualização e a conteinerização emergindo como duas das principais para modelagem dos nós. Além dos benefícios em termos de segurança, essas abordagens permitem um compartilhamento mais eficiente de recursos, eliminando a necessidade de reservar um nó físico para cada experimento. Outro aspecto do isolamento abordado pelos *testbeds* diz respeito à segmentação da rede, onde soluções baseadas em segmentação dividem as interfaces do

caminho de dados de um experimento das interfaces do plano de controle (FIT IoT, GENI, dentre outros). A execução segura é outro aspecto relacionado ao isolamento, empregando as mesmas ferramentas mencionadas anteriormente, mas com uma ênfase diferente. As principais características dos *testbeds* apresentados são resumidas na Tabela 5.1.

Tabela 5.1: Comparação dos *testbeds* apresentados

Testbed	**Recursos**	**Flexibilidade**	**Escala**	**Execução Segura**	**Disponibilidade**
DeterLab (Antigo)	CT, VM, *Phy**	Alta	Alta	Sim	Educação apenas
DeterLab (MergeTB)	VMs, *Phy*	Alta	ND	Sim	Sim
FIT-IoT LAB	*Phy*	Alta	Alta	ND	Sim
Gotham	CT e VMs	Alta	Média	Sim	Sim (código fonte)
GENI	VM, CT, *Phy*	Alta	Alta	Sim	Descontinuado
FABRIC	VMs	Alta	Alta	Sim	Sim
Fed4Fire+	Misto‡	Misto‡	Misto‡	Misto‡	Sim
TON_IoT	*Phy* e VMs	Baixa	Baixa	ND	Apenas *dataset*
Bot-IoT	VMs	Baixa	Baixa	ND	Apenas *dataset*

Phy Dispositivos físicos; *VM* Máquinas Virtuais; *CT* Containers; ‡ Dependente do *testbed* federado selecionado; *ND* Não detalhado; $*$ Dentre outros

Além dos *testbeds* anteriormente discutidos, há uma variedade de outros *testbeds* na literatura, cada um com focos de atuação distintos e arquiteturas diferentes. Por exemplo, em Koroniotis, Moustafa, Schiliro et al. (2021), é apresentado um *testbed* voltado para o estudo de dispositivos IoT industriais (IIoT). Outros *testbeds*, como o abordado em Siboni et al. (2018), realizam análises das vulnerabilidades de dispositivos IoT. Em Thom et al. (2021), é apresentado um *testbed* que modela uma cidade, incluindo dispositivos IIoT e IoT, juntamente com tecnologias como redes definidas por software (SDN) para experimentação. Além disso, vale destacar, a influência de *testbeds* como o Planetlab (CHUN et al., 2003) e Emulab, que desempenharam um papel significativo na formação do panorama atual de *testbeds* existentes.

Por fim, é importante ressaltar que os *testbeds* destinados à pesquisa experimental em cibersegurança são ambientes complexos e dispendiosos, não apenas em termos financeiros, mas também em relação ao tempo necessário para sua criação, manutenção, atualização e validação. Por estes e outros motivos, muitos *testbeds* são descontinuados ou substituídos ao longo do tempo, ou até mesmo nunca são disponibilizados para uso por pesquisadores externos.

5.3. MENTORED *Testbed*

O MENTORED *Testbed* é uma infraestrutura avançada projetada para conduzir experimentos em cibersegurança, com um enfoque específico na análise de vulnerabilidades, ataques e estratégias de defesa associadas aos dispositivos da Internet das Coisas (IoT, *Internet of Things*). Esse *testbed* se destaca pela sua capacidade de criar ambientes de rede realistas, permitindo que pesquisadores avaliem a eficácia de diferentes técnicas de

segurança em um ambiente controlado, mas ao mesmo tempo complexo e diversificado.

A arquitetura do MENTORED *Testbed*, conforme delineado por Gemmer et al. (2023), destaca-se por sua flexibilidade e escalabilidade, o que facilita a integração com uma gama de dispositivos IoT. Um dos seus principais diferenciais é sua independência em relação aos recursos de processamento, o que possibilita a construção e adaptação do *testbed* de acordo com as necessidades específicas de cada experimento. Além disso, a arquitetura oferece suporte a uma diversidade de funcionalidades que simplificam as diferentes etapas do ciclo de vida dos experimentos, desde a sua definição e execução até a análise e interpretação dos resultados (MEYER; GEMMER; SCHWARZ et al., 2022).

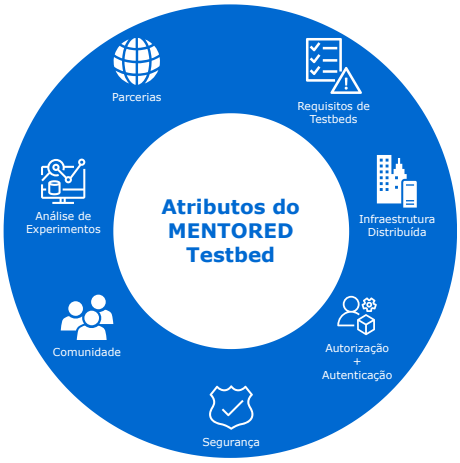


Figura 5.3: Atributos do MENTORED *Testbed*. Adaptado de (GEMMER et al., 2023).

A Figura 5.3 ilustra os principais atributos do MENTORED *Testbed*, os quais são detalhados a seguir:

- Parcerias: o MENTORED *Testbed* promove colaborações com diferentes projetos e equipes, visando a melhoria contínua da infraestrutura. Por exemplo, uma equipe da RNP está atualmente envolvida no desenvolvimento do *testbed*, especialmente para aprimorar a infraestrutura baseada em Kubernetes;
- Requisitos de *testbeds*: o *testbed* foi construído e é atualizado de forma a atender os requisitos essenciais para *testbeds* de cibersegurança. Estes requisitos incluem fidelidade, flexibilidade, escalabilidade, isolamento, execução segura, reprodutibilidade, transparência, perspectiva centrada no usuário e acesso em tempo real;
- Infraestrutura distribuída: ao contrário das simulações executadas em uma única máquina, a infraestrutura do MENTORED *Testbed* se destaca por sua distribuição entre diversos nós. Essa abordagem possibilita uma representação mais realista da comunicação entre os diferentes componentes da topologia, refletindo com maior precisão as dinâmicas das redes complexas;
- Autenticação e Autorização: o *Testbed* implementa autenticação federada e mecanismos robustos de autorização para garantir que apenas usuários autorizados possam acessar seus recursos. Essa abordagem visa reduzir o atrito no acesso aos recursos do *testbed*, tornando o processo mais fluido para o usuário;

- Segurança: devido à criticidade de experimentos científicos de cibersegurança, é fundamental que o MENTORED *Testbed* consiga evitar que os experimentos conduzidos sejam direcionados para a Internet e que não ocorra vazamento de dados entre diferentes experimentos, independentemente de serem executados por um mesmo usuário ou por usuários diferentes;

- Comunidade: todos os usuários do MENTORED *Tesbed* podem contribuir por meio de *feedbacks* e compartilhando suas definições de experimentos, possibilitando que outros pesquisadores repliquem e validem os resultados obtidos;

- Análise de experimentos: o MENTORED *Testbed* oferece aos usuários acesso contínuo aos dados e resultados dos seus experimentos, tanto em tempo real durante sua execução quanto após sua conclusão. Isso é viabilizado por meio de um mecanismo de armazenamento que otimiza a seleção de arquivos para preservação futura e consultas. Adicionalmente, para cada execução de experimento são gerados arquivos de *logs* detalhados, fundamentais tanto para a análise minuciosa dos experimentos quanto para a identificação e resolução de eventuais falhas.

O MENTORED *Testbed* incorpora tecnologias avançadas de virtualização e conteinerização, como Docker e Kubernetes. Isso permite a criação de ambientes de teste isolados e reproduzíveis, uma característica crucial para a análise de *malwares* e ataques de rede, onde a contenção e a repetibilidade dos experimentos são essenciais para garantir a validade dos resultados. Além disso, o *testbed* provê suporte a uma ampla gama de ferramentas de análise de segurança e monitoramento de rede, proporcionando aos usuários a capacidade de detectar, analisar e responder a incidentes de segurança em tempo real.

Em resumo, o MENTORED *Testbed* oferece uma plataforma avançada e flexível para a investigação de ameaças de segurança. Com sua arquitetura escalável, interface de usuário amigável e integração de tecnologias avançadas, o *testbed* facilita a execução de experimentos complexos, o que pode contribuir significativamente para o avanço da pesquisa e desenvolvimento em cibersegurança.

5.3.1. MENTORED *framework*

Devido à complexidade envolvida no desenvolvimento de *testbeds* que atendam aos requisitos apresentados na Seção 5.1.2, Gemmer et al. (2023) propuseram um *framework* genérico que serve de referência para construção de *testebeds* para experimentação em cibersegurança, incluindo ataques DDoS provenientes de dispositivos de IoT. Qualquer desenvolvedor pode instanciar esse *framework* para definir seu próprio *testbed*, usando as tecnologias que desejar. Esta seção detalha como utilizar este *framework* para a experimentação em cibersegurança, descrevendo as entidades e os módulos envolvidos e como eles se relacionam para viabilizar a definição, execução e análise de experimentos.

De acordo com Gemmer et al. (2023), o *framework* proposto, usado construção do MENTORED *Testbed*, considera um controlador geral, que é responsável por interligar as cinco principais entidades: 1) Provedor de recursos de IoT; 2) Provedor de recursos de processamento; 3) Canal de comunicação de recursos; 4) Provedor federado de identidade; 5) Orquestrador e gerenciador de recursos para experimentos (*Master*).

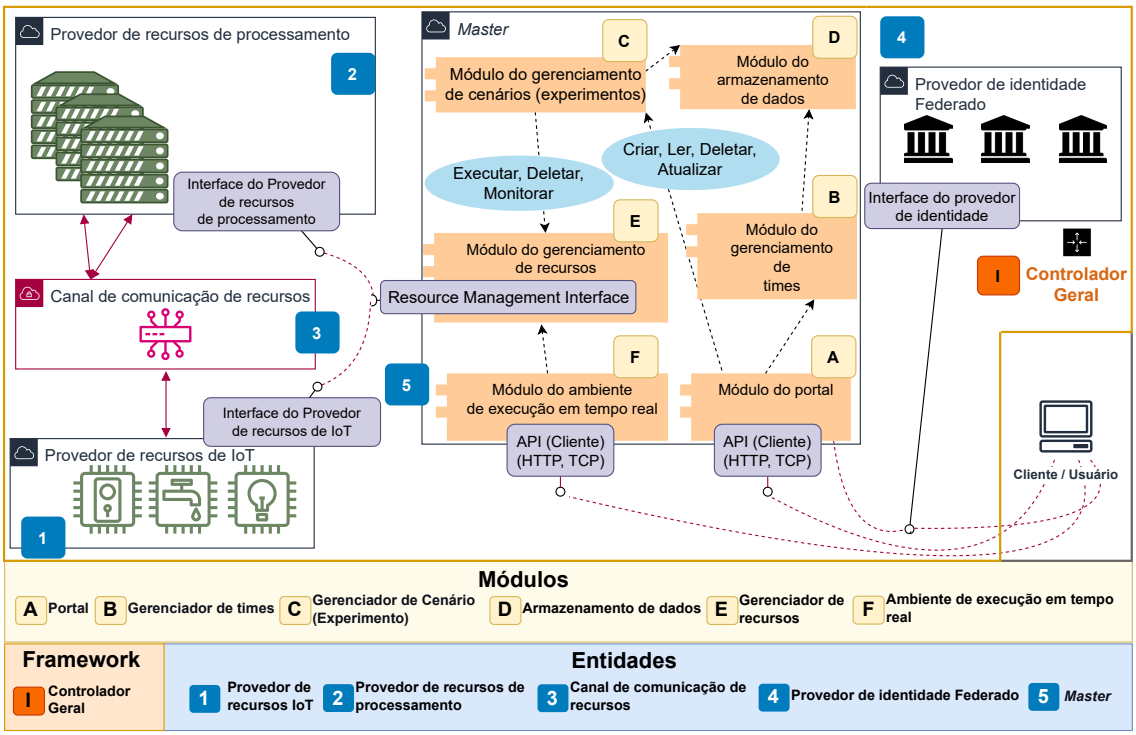


Figura 5.4: *Framework* utilizado pelo MENTORED *Testbed*. Adaptado de (GEMMER et al., 2023).

A Figura 5.4 apresenta os principais componentes do *framework* usado pelo MENTORED *Testbed* e suas relações. O *framework* considera como premissa a distribuição geográfica de provedores de recursos de processamento que são utilizados para executar experimentos. O *framework* considera ainda a existência de provedores específicos para recursos IoT, os quais podem se comunicar com os demais provedores, por meio de um canal de comunicação implementado exclusivamente para o *testbed*.

O *Master* é a entidade do *framework* que gerencia a intermediação entre experimentadores e os provedores de recursos disponíveis no *testbed*. Após se autenticar por meio de um provedor de identidade federado e obter a devida autorização, o usuário seguindo as etapas descritas na Figura 5.5 poderá ter acesso a diversas funcionalidades do *Master* como a definição, execução, armazenamento, coleta de dados e acesso em tempo real a qualquer recurso que faça parte do experimento. Um usuário autenticado poderá interagir com o *Master* por meio de duas interfaces: um portal *web*, que consiste em uma interface gráfica amigável; e uma *API* REST, que permite a interação programável com o *testbed* por meio do protocolo HTTP.

5.3.2. Requisitos e estratégias do MENTORED *Testbed*

Assim como outros *testbeds* apresentados na Seção 5.2, o MENTORED *testbed* também foi desenvolvido para atender diferentes requisitos para que o experimentador possa definir, executar e analisar os seus experimentos da melhor forma possível. A Figura 5.6 ilustra a relação entre os requisitos considerados pelo MENTORED *testbed* e as estratégias utilizadas para suprir esses requisitos.

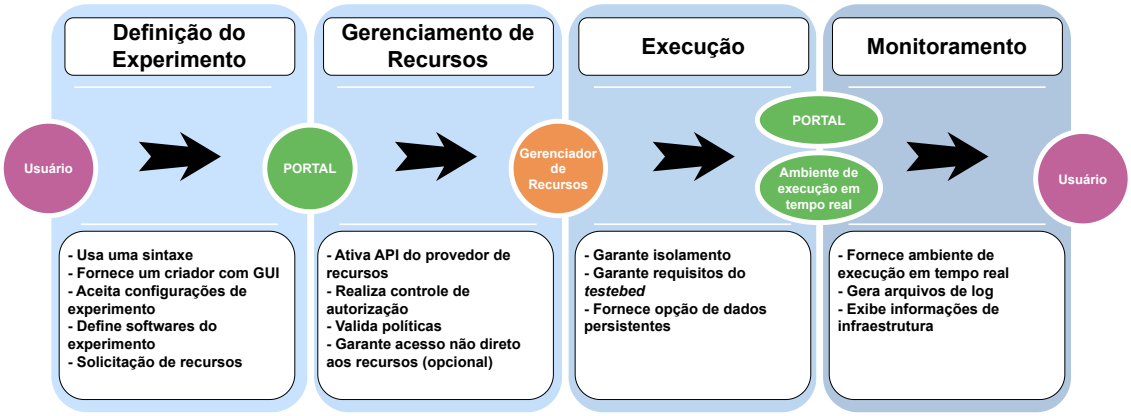


Figura 5.5: Fluxo de execução de experimento no MENTORED *Framework*.

5.3.2.1. Reprodutibilidade

Um dos requisitos considerados no MENTORED *testbed* é a reprodutibilidade de experimentos. Esse requisito é atendido por meio da padronização de definições de experimentos, os quais são feitos por meio de um arquivo do tipo YAML que estende a definição dos recursos *pods* na tecnologia Kubernetes[32].

No Kubernetes, um *pod* pode ser definido especificando um ou mais contêineres Docker e os parâmetros para inicialização (privilégios do contêiner, variáveis de ambiente, comandos para inicialização do contêiner, etc.). Os contêineres Docker por sua vez são definidos utilizando *Dockerfiles*[33], especificando o equivalente a sistema operacional como base e comandos que podem ser usados para instalar e executar softwares.

Os experimentos definidos no MENTORED *testbed* devem conter os seguintes elementos: nome do experimento, tempo limite para a execução do experimento, lista de *pods* (seguindo sintaxe Kubernetes) e tipo de topologia utilizada no experimento. Um recurso importante do *testbed* é a possibilidade de criar réplicas de *pods*, de forma que o experimentador possa escalar o experimento conforme desejar. Em uma atualização futura, o MENTORED *testbed* terá também um repositório de experimentos e definições de *pods*, potencializando a cooperação de pesquisadores, por meio do compartilhamento e reprodutibilidade de definições experimentos.

Um recurso importante em *testbeds* de cibersegurança é a possibilidade de definir topologias de redes personalizadas para diferentes tipos de experimentos. Devido à modularidade dos recursos de processamento no MENTORED *testbed*, o experimentador sempre poderá ter a opção de definir em qual recurso físico cada *pod* de seu experimento será executado. Esse tipo de recurso é facilitado pelo Kubernetes e a infraestrutura definida por software do *Cluster* Nacional da RNP.

[32]https://kubernetes.io/docs/concepts/workloads/pods/
[33]https://docs.docker.com/get-started/02_our_app/

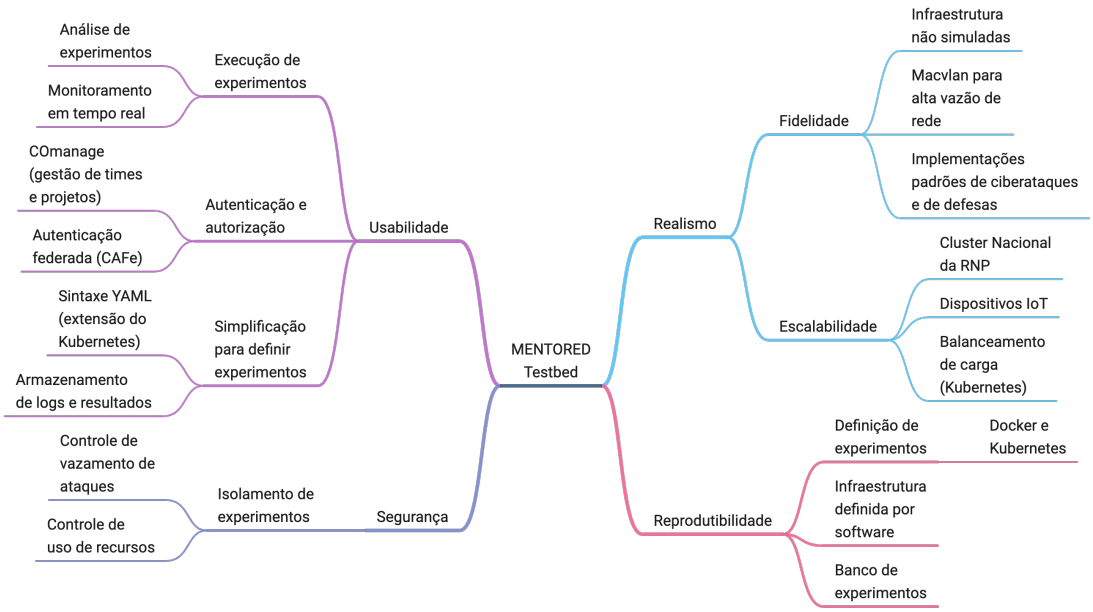


Figura 5.6: Requisitos e estratégias consideradas no MENTORED *testbed*.

5.3.2.2. Realismo

Embora seja impossível alcançar total realismo em muitos cenários de ambientes controlados para experimentos, muitos *testbeds*, incluindo o MENTORED, tem como objetivo serem o mais realista possível. O requisito de realismo pode ser dividido em fidelidade e escalabilidade. No MENTORED *testbed*, a fidelidade é alcançada usando múltiplos recursos físicos de processamento para executar experimentos, em contraste com alternativas como o *Network Simulator* (GOMEZ et al., 2023). Outro recurso importante é o uso da tecnologia de rede Macvlan. O Macvlan é uma tecnologia que reduz os componentes virtuais necessários para comunicar *pods* em um *cluster* Kubernetes, consequentemente maximizando assim o desempenho e aproximando-se da capacidade real dos recursos de comunicação e processamento (CLAASSEN; KONING; GROSSO, 2016).

Dentro do repositório de definições de experimentos do MENTORED *testbed*, estarão incluídos diversos contêineres Docker com implementações de elementos comumente necessários para experimentos de cibersegurança, como softwares para simular ataques DDoS e servidores *Web*. O experimentador poderá utilizar esses softwares, e outros caso deseja implementá-los, e replica-los em diferentes nós do *Cluster* Nacional da RNP. Além disso, em breve será possível utilizar também dispositivos IoT que se comunicarão com os nós de processamento do *Cluster* Nacional. A orquestração e balanceamento de carga da execução e comunicação dos contêineres são realizados em alto nível pelo MENTORED *Master* e em baixo nível pelo Kubernetes.

5.3.2.3. Segurança

Para que o ambiente seja seguro para o experimentador, o MENTORED *testbed* implementa estratégias que asseguram o isolamento dos experimentos. Isso significa que todas as atividades realizadas pelos softwares dos experimentos são controladas para não interferir no funcionamento adequado do *cluster*, nem na execução de outros experimentos. Além disso, o *Kubernetes* pode ser usado para evitar que softwares potencialmente maliciosos tenham acesso à internet e infectem ou ataquem serviços não relacionados aos experimentos (BUDIGIRI et al., 2021; NGUYEN et al., 2020).

5.3.2.4. Usabilidade

Conforme discutido anteriormente, uma das estratégias principais do MENTORED *testbed* é simplificar o processo de definição de experimentos. Os usuários podem criar experimentos complexos utilizando apenas um arquivo no formato YAML, estendendo a sintaxe do Kubernetes, conhecido por sua ampla popularidade e documentação (SAYFAN, 2017). Na definição dos experimentos, os experimentadores podem especificar quais arquivos e *logs* devem ser salvos após a conclusão do experimento, facilitando análises posteriores.

Para definir e executar experimentos no MENTORED *testbed*, o usuário deve autenticar-se e obter a autorização adequada. Essas etapas são implementadas usando as tecnologias de autenticação federada CAFe e de gerenciamento de times COmanage, o que será explicado com mais detalhes na Seção 5.3.3.3.

Após receber a devida autorização, um experimentador poderá executar a definição de um experimento dentro do contexto de um projeto. Se o projeto tiver acesso aos recursos especificados na definição do experimento, um novo ciclo de vida de experimento é iniciado. O ciclo de vida do experimento será explicado com mais detalhes na Seção 5.3.3.4. Durante esse processo, o experimentador pode acessar em tempo real qualquer *pod* em execução por meio da tecnologia *Webkubectl*[34]. Isso permite ao experimentador acompanhar os experimentos e realizar as análises necessárias dos dados gerados no ambiente do *testbed*.

5.3.3. Recursos tecnológicos do MENTORED *Testbed*

O desenvolvimento do MENTORED *testbed* é uma combinação de diversas tecnologias, sendo algumas delas criadas exclusivamente para o *testbed*, como ilustrado na Figura 5.7. O MENTORED *testbed* é uma instância do *framework* apresentado na Figura 5.4 que utiliza o *cluster* nacional da RNP como principal infraestrutura para os provedores de recursos de processamento. A manutenção e acesso desses dispositivos são realizados majoritariamente usando a tecnologia Kubernetes. Atualmente, a versão v1.27.7 do Kubernetes é utilizada tanto pelo MENTORED quanto pelo *cluster* nacional da RNP.

[34]https://github.com/1Panel-dev/webkubectl

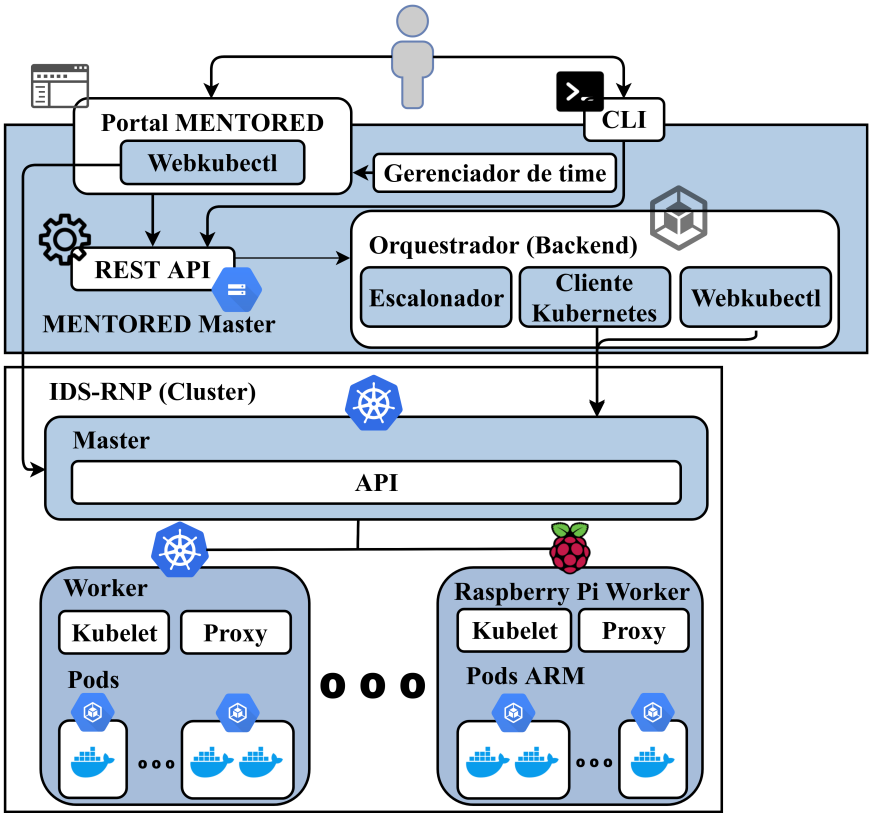


Figura 5.7: Implementação do MENTORED *testbed framework*. Adaptado de (GEMMER et al., 2023).

5.3.3.1. MENTORED Master

O MENTORED *Master* é a principal entidade no MENTORED *Testbed*, responsável por intermediar o experimentador e os recursos computacionais. Sua implementação é feita utilizando a ferramenta Docker-Compose, que gerencia diversas tecnologias que compõe o MENTORED *Master*. A entidade MENTORED *Master* pode ser dividida nos seguintes módulos:

- **Portal web**: interface GUI usada pelo experimentador, implementada usando o *framework* React.
- **Módulo de gestão de times**: implementado usando a tecnologia COmanage, este controla a interação entre diferentes experimentadores e suas autorizações para acessar determinados recursos do *testbed*.
- **MENTORED *core***: responsável por implementar o gerenciamento de recursos e comunicação com os provedores de recursos. Sua implementação é feita em Python, e utiliza a biblioteca oficial do Kubernetes para gerenciar *pods* e outros recursos dos provedores de recurso de processamento. Além dos recursos padrões do Kubernetes, o MENTORED *core* implementa diversas funcionalidades importantes para experimentos, tais como: mecanismos de *warmup* (sincronização da ini-

cialização), gerenciamento de *logs*, armazenamento de arquivos gerados por *pods*, definição de IPs e mecanismos de comunicação entre *pods*.

- **MENTORED *api***: serviço *web* responsável por validar e armazenar dados relacionados a experimentos. Implementa uma *api* REST usando o *framework* Django e biblioteca Django-Rest, que permite a adoção da especificação OpenAPI. A *api* implementada é utilizada pelo Portal *web*, de forma que os recursos implementados pelo MENTORED *core* possam ser acionados usando o protocolo HTTP. Atualmente, não existe nenhuma biblioteca oficial em Python ou qualquer outra linguagem para acessar o MENTORED *api*, mas o experimentador pode utilizar a documentação da *api* para utilizar o MENTORED *testbed* de forma automatizada e programável.
- **Módulo do ambiente de execução em tempo-real**: implementado usando a tecnologia Webkubectl, que permite utilizar comandos do Kubernetes por meio de *web browsers*. Seu uso é controlado por mecanismos de acesso via *tokens* gerados pelo Kubernetes, que garantem isolamento de experimentos por meio do recurso chamado *namespace*. Um usuário pode acessar qualquer *pod* em execução dentro dos *namespaces* relacionados aos experimentos que este tem acesso. Uma vez que um usuário tem acesso a um *pod*, qualquer comando pode ser executado dentro do contêiner em execução, semelhante a um acesso SSH.

5.3.3.2. Portal do MENTORED *testbed*

O portal que implementa a principal interface de acesso de experimentadores ao MENTORED *testbed* consiste em uma aplicação *web* que acessa a API *REST* disponibilizada pelo MENTORED *master*. Por meio do acesso a essa API, o usuário pode ter acesso a diversas funcionalidades e recursos do *testbed*, como no *Dashboard* ilustrado na Figura 5.8a. Além disso, o usuário pode gerenciar os projetos ao qual pertence como mostrado na Figura 5.8b[35]. Por fim, definições e execuções experimentos de experimentos também podem ser criados e monitorados no portal, assim como demonstra as Figuras 5.11a e 5.11b.

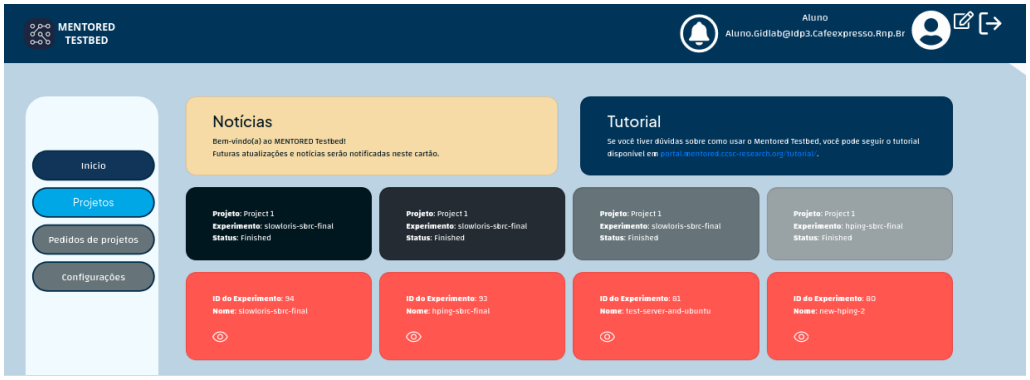
5.3.3.3. Gestão de projetos e times

Um projeto de pesquisa em cibersegurança geralmente envolve a colaboração de uma equipe multidisciplinar, composta por um coordenador principal do projeto, pesquisadores e desenvolvedores. Para garantir que todos os membros da equipe possam contribuir efetivamente, é essencial que haja uma estrutura clara de responsabilidades e permissões, que permita a distribuição das tarefas e a coordenação das atividades.

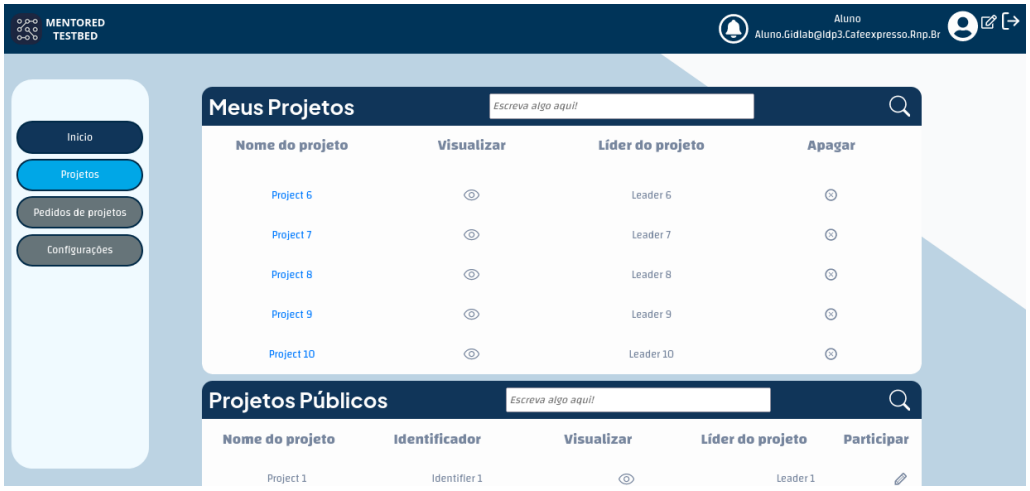
O COmanage[36] é uma ferramenta versátil que pode ser incorporada em soluções de gerenciamento de identidades e colaboração (FLANAGAN et al., 2012). O MENTORED *Master* faz uso do COmanage para gerenciar os usuários e suas permissões, facili-

[35]O recurso de gerenciamento de times ainda está em desenvolvimento no MENTORED *testbed*

[36]https://incommon.org/software/comanage



(a) *Dashboard* do portal no MENTORED *testbed*.



(b) Interface para gestão de times (em desenvolvimento).

Figura 5.8: Portal do MENTORED *testbed*.

tando assim a colaboração nos experimentos de cibersegurança usando o MENTORED *Testbed*. Para a integração do COmanage com o MENTORED *Testbed* foi modelado um conjunto de papéis que podem ser atribuídos aos usuários, determinando os recursos e funcionalidades disponíveis para cada usuário de acordo com o seu papel.

Na Figura 5.9 é apresentado um diagrama de casos de uso do MENTORED *Testbed*. O atores são os usuários do sistema, de acordo com os papéis que desempenham, enquanto os casos de uso representam as funcionalidades disponíveis no sistema, para cada ator.

O ator "Não membro" representa usuários que não estão cadastrados no sistema. Assim, a única interação possível é a realização do cadastro no sistema. Todo pedido de cadastro é avaliado por um Administrador do *testbed*, que pode aceitar ou rejeitar a solicitação.

O ator "Membro ativo" representa usuários que estão cadastrados e podem requisitar a criação de novos projetos. O pedido de criação de um novo projeto é avaliado por um Administrador do *testbed*, que pode aceitar ou rejeitar a solicitação. Uma vez que o projeto é aceito, o usuário que fez a requisição se torna o Líder de Projeto. Trata-se do

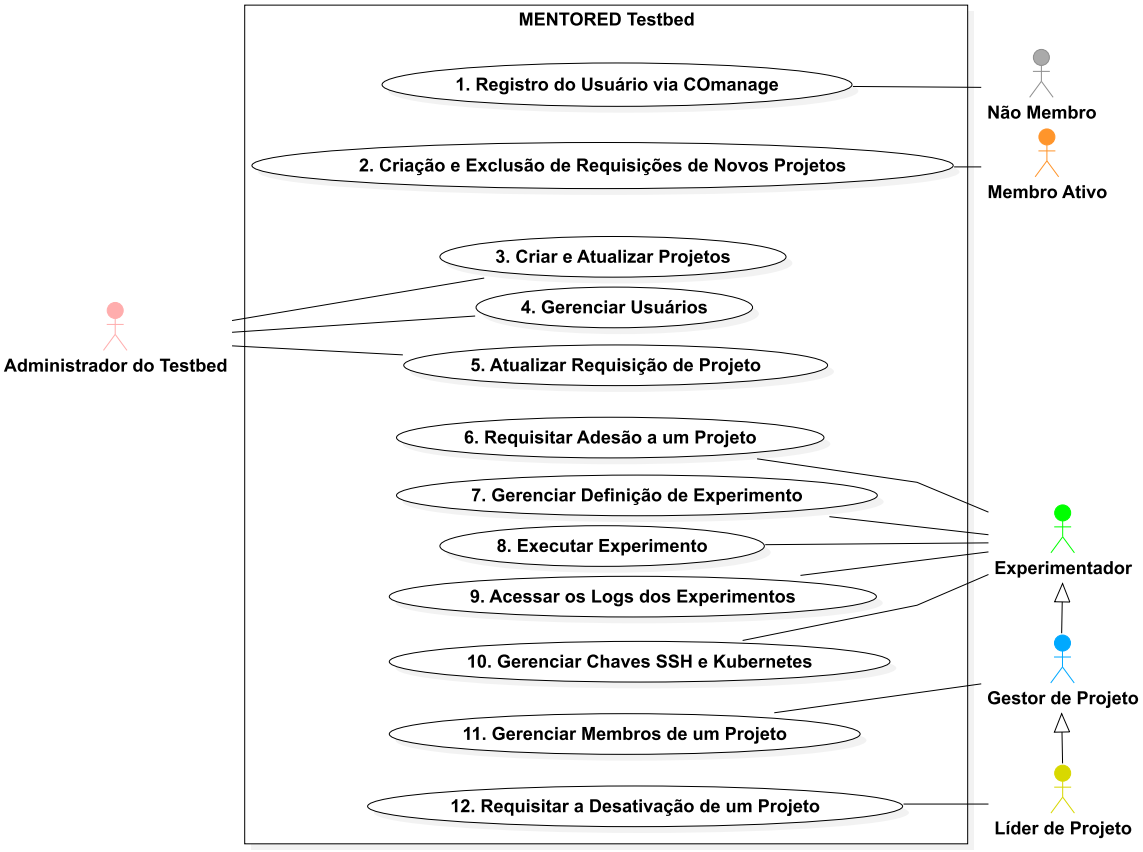


Figura 5.9: Diagrama de Casos de Uso do MENTORED *Testbed*.

papel com maior privilégio dentro de um projeto, possuindo os mesmos direitos concedidos aos papéis Gestor de Projeto e Experimentador. Compete ao Líder de Projeto delegar os papéis de Gestor de Projeto e Experimentador aos usuários do projeto

O "Gestor de Projeto" é um papel que pode ser designado a mais de um usuário que ficará responsável pela administração de usuários e experimentos. Ou seja, o Gestor de Projeto pode gerenciar membros da equipe e configurar experimentos. Por fim, o "Experimentador" é o usuário que pode realizar experimentos dentro do projeto ao qual está associado.

Cabe ressaltar que uma vez que o pesquisador possua um usuário no sistema, esse poderá participar de diversos projetos, podendo assumir diferentes papéis em cada um deles. Dessa forma, o MENTORED *Testbed* oferece uma estrutura flexível e adaptável, que permite a colaboração eficiente entre os membros da equipe.

O gerenciamento de equipes[37] traz diversas funcionalidades, como o registro de usuários pelo COmanage que simplifica o processo de entrada. Após logado no sistema, o usuário pode criar e remover requisições de novos projetos diretamente na plataforma. Além disso, a administração de membros e permissões em projetos gerencia os papéis atribuídos a cada membro. Essas funcionalidades visam fomentar uma colaboração mais

[37]A implementação do gerenciamento de equipes e suas funcionalidades encontra-se em desenvolvimento no momento.

efetiva e segura entre os membros da equipe. Adicionalmente, os usuários possuem a oportunidade de conduzir e supervisionar experimentos, acessar registros e gerenciar chaves de autenticação SSH e configurações do Kubernetes, com o objetivo de assegurar a segurança e a eficácia do sistema.

Essas funcionalidades proporcionam maior controle e flexibilidade aos usuários durante sua interação com o sistema, ao mesmo tempo em que simplificam a administração de projetos e experimentos. Com isso, o objetivo da gestão é proporcionar uma experiência mais satisfatória para todos os envolvidos no MENTORED *Testbed*.

5.3.3.4. Ciclo de vida do experimento

Conforme ilustrado na Figura 5.10, o ciclo de vida de um experimento pode ser dividido em três etapas: 1) Definição do experimento, onde o experimentador define topologias de rede, softwares e outros recursos que são especificados em uma sintaxe definida pelo MENTORED *Master*; 2) A execução do experimento, onde um experimentador seleciona uma definição de experimento e o momento em que esse experimento será iniciado; 3) As análises dos experimentos, que incluem armazenamento e processamento de *logs*, alertas e, se necessário, o monitoramento dos experimentos em tempo real.

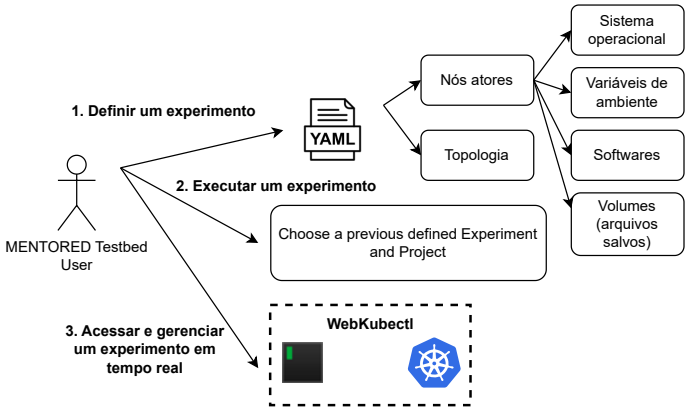
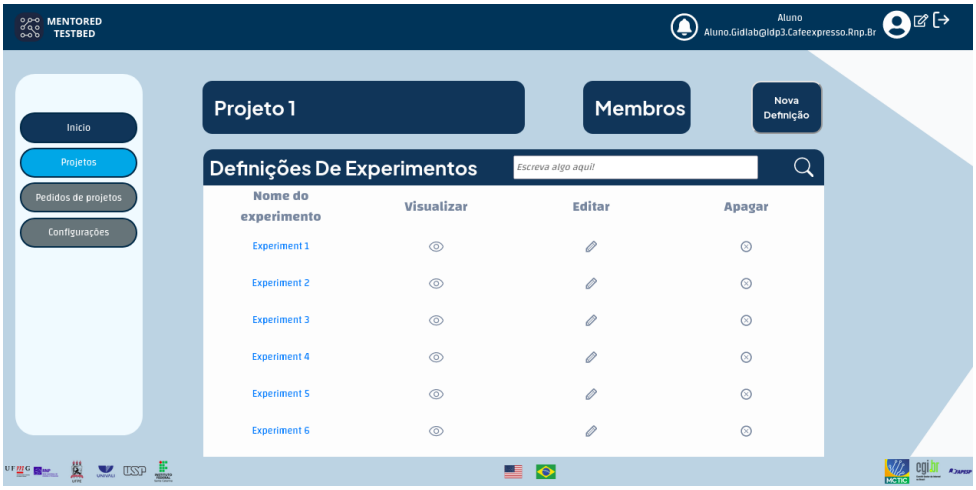


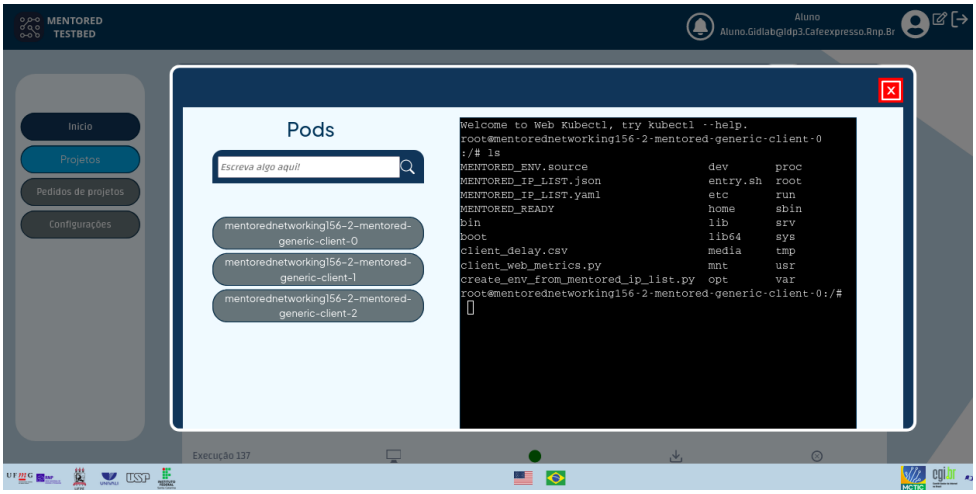
Figura 5.10: Perspectiva do usuário para definição de um experimento no MENTORED *testbed*.

A Seção 5.4, discutirá exemplos de uso práticos do ciclo de vida de experimentos, usando o MENTORED *Master* e o seu portal para executar as etapas do ciclo, considerando a perspectiva do experimentador. A Figura 5.11a ilustra um exemplo da página de definição e controle de experimentos e a Figura 5.11bilustra o monitoramento em tempo real de experimentos no Portal do MENTORED *Testbed*.

De forma complementar, em Meyer, Gemmer, Santana et al. (2024), os autores descrevem um fluxo para criação e análise de *datasets* representativos de ataque de negação de serviço, usando o MENTORED Testbed, conforme resumido na Figura 5.12.



(a) Página para controle de definições de experimentos.



(b) Página para monitoramento em tempo real de experimentos.

Figura 5.11: Páginas sobre experimentos do Portal do MENTORED *testbed*.

5.3.3.5. Aprimoramentos do MENTORED *Testbed*

A estrutura e funcionalidade do MENTORED *Testbed* têm sido submetidas a análises e melhorias contínuas, conforme documentado em diversos estudos anteriores (PRATES JR et al., 2021; MEYER; GEMMER; SCHWARZ et al., 2022; GEMMER et al., 2023). A infraestrutura do *testbed* está em constante evolução, focando não apenas na correção de falhas, mas também na introdução de novas funcionalidades que aprimoram sua eficácia e usabilidade.

Recentemente, uma significativa atualização do MENTORED *Testbed* facilitou o desenvolvimento de um procedimento eficiente para a criação de *datasets* dedicados a pesquisas de cibersegurança (MEYER; GEMMER; SANTANA et al., 2024). As inovações que estão em fase de desenvolvimento e prometem impulsionar a capacidade do *testbed* incluem:

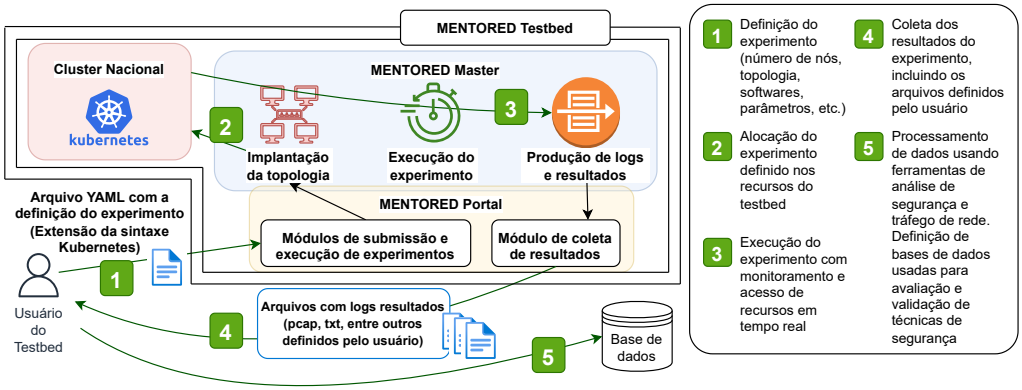


Figura 5.12: Fluxo que usa o MENTORED *testbed* para criação e análise de *datasets*. Fonte: (MEYER; GEMMER; SANTANA et al., 2024).

- **Aprimoramento na coleta de *logs* e resultados**: Os experimentos de cibersegurança no *testbed* podem envolver múltiplos nós operando softwares de maneira isolada, o que gera uma grande quantidade de dados e *logs*. Por exemplo, a simulação de ataques DDoS pode resultar na produção de *logs* de monitoramento de rede que alcançam a ordem de *gigabytes* em poucos minutos. Estão sendo desenvolvidos mecanismos avançados de monitoramento que permitirão aos usuários acompanhar os experimentos e acessar os resultados de maneira simplificada e ágil. Os principais desafios desta frente incluem a compressão eficaz de dados e a coleta de *logs* de diferentes componentes do MENTORED (como Kubernetes, softwares emulados e o MENTORED *Master*).

- **Desenvolvimento de um repositório de definições de experimentos e imagens Docker**: A efetivação de testes e a validação das capacidades do *testbed* são conduzidas através de experimentos detalhadamente definidos. Essas definições muitas vezes requerem a implementação de configurações essenciais para a execução de testes de cibersegurança, tais como a sincronização de nós e a configuração de serviços e ataques. O acervo de imagens Docker está em constante expansão e futuramente permitirá que o *testbed* ofereça funcionalidades para a publicação e compartilhamento de configurações experimentais com a comunidade de usuários.

- **Gestão de recursos e autorizações**: Conforme mencionado anteriormente, o MENTORED *Testbed* incorpora diversas tecnologias para autenticação e autorização, além do gerenciamento de equipes. Um aspecto crucial atualmente em desenvolvimento é a implementação de um sistema robusto para o controle de recursos utilizados por cada experimentador, possibilitando a definição precisa da quota de recursos disponíveis para cada equipe ou usuário individual nos seus experimentos.

- **Dispositivos IoT**: Atualmente a equipe do MENTORED *Testbed* está realizando análises para a inclusão de dispositivos IoT para experimentação. Em específico, dispositivos Raspberry Pi 4 (4G) estão sendo utilizados em um *cluster* IoT dedicado, realizando provas de conceito e validações do funcionamento da pilha de *software* utilizada pelo MENTORED *Testbed*. Ademais, foi obtido sucesso em testes preliminares de conectividade WiFi, isto é, servindo como a interface de rede

para nós modelados em um experimento. Como próximos passos, serão realizados estudos da integração desdes dispositivos com o Cluster Nacional RNP, possibilitando então a criação de experimentos abrangendo dispositivos x86 tradicionais e dispositivos IoT, simultaneamente.

5.4. Estudos de casos

Com o intuito de demonstrar a capacidade do MENTORED *Testbed*, esta seção apresenta dois estudos de caso que abordam a execução de experimentos práticos com foco em ataques DDoS[38]. As atividades práticas para este minicurso foram divididas em duas partes. A primeira atividade consiste em um ataque volumétrico utilizando a ferramenta hping3 (veja Seção 5.4.1), enquanto a segunda atividade apresenta um cenário em que são explorados problemas que servidores *Web* podem enfrentar ao lidar com milhares de conexões simultâneas, por meio de um ataque com o *Slowloris* (veja 5.4.2).

Ambas as atividades práticas são compostas pelo mesmo número de clientes, atacantes, vítima, dinâmica de ataque e tempo de execução, variando apenas o tipo de ataque e a distribuição geográfica. Como ilustrado na Figura 5.13, o ataque tem uma duração total de 300 segundos, dividido em três partes: 1) O pré-ataque ocorre nos primeiros 60 segundos, composto apenas pelo tráfego dos clientes legítimos; 2) O ataque é iniciado aos 60 segundos e tem uma duração de 180 segundos, agregando o tráfego legítimo dos clientes ao tráfego malicioso dos atacantes com o objetivo de interromper o serviço, e; 3) O período pós-ataque é composto pelos últimos 60 segundos, onde é avaliado se os clientes legítimos ainda conseguem estabelecer a conexão com o servidor *Web*.

Figura 5.13: Cronograma do Ataque.

Como descrito na Seção 5.3.2.1, a definição de um experimento no MENTORED *testbed* é realizada por meio de um código `YAML` que estende objetos do Kubernetes, acrescentando assim características exclusivas do MENTORED *testbed*. Para ambos os estudos de casos que serão apresentados nas atividades práticas, a definição do experimento foi dividida em três partes: 1) Definição do *pod* que atuará como vítima; 2) Definição dos *pods* responsáveis pelo ataque, e; 3) definição dos *pods* que atuarão como clientes legítimos e que farão requisições ao serviço disponibilizado pelo *pod* vítima.

Na Listagem 5.1 é apresentada a definição dos *pods* que atuarão como clientes legítimos que farão requisições HTTP GET ao servidor *web* hospedado no *pod* vítima. É

[38]Uma documentação e código-fonte dos softwares utilizados nos experimentos está disponível em: https://github.com/mentoredtestbed/minicurso-sbrc-2024-testbeds

possível observar que serão instanciadas 70 réplicas do cliente, cada um com uma imagem personalizada responsável por efetuar as requisições *Web* em um intervalo de 1 segundo durante 300 segundos. O tempo de resposta das requisições é salvo em um arquivo do tipo *csv*. Por fim, esse conjunto de clientes será executado na região de Goiás do *Cluster* Nacional da RNP.

```yaml
- name: 'generic-client-go'
  persitent_volume_path: "/client_delay.csv"
  replicas: 70
  containers:
    - name: 'client-go'
      image: ghcr.io/mentoredtestbed/generic-client:latest
      command: ["/entry.sh"]
      args: ['python3', 'client_web_metrics.py', "1", "1"]
      env:
        - name: TIMEOUT_CMD
          value: "300"
        - name: ADD_SERVER_IP_TO_COMMAND
          value: "true"
      resources:
        requests:
          memory: "64Mi"
          cpu: "100m"
        limits:
          memory: "128M"
          cpu: "200m"
  region: 'ids-go'
```

Listagem 5.1: YAML descrevendo o cliente *Web*.

Na Listagem 5.2 é apresentada a definição dos *pods* que atuarão como atacantes, responsáveis por realizar o ataque DDoS. É possível perceber que a definição do atacante é bastante semelhante à definição do cliente, com a diferença que é feito uso de uma imagem de contêiner configurada com os ataques utilizados. Sendo assim, é possível determinar que serão utilizados 10 atacantes com a ferramenta *slowloris* na porta 80. O ataque será iniciado aos 60 segundos e interrompido aos 180 segundos. Por fim, esse conjunto de atacantes será posicionado na região da Paraíba do *Cluster* Nacional da RNP.

5.4.1. Atividade prática I: ataque volumétrico com hping3

Nessa atividade será conduzido um ataque DDoS volumétrico por meio da ferramenta hping3[39]. O hping3 é uma ferramenta de código aberto amplamente utilizada para testes de penetração e avaliação de redes, porém também pode ser empregada de forma maliciosa em ataques DDoS. Uma das características do hping3 é a sua capacidade de enviar pacotes ICMP, UDP e TCP com diferentes *flags* e opções, o que possibilita a simulação de diversos tipos de tráfego malicioso. Além disso, o hping3 permite a falsificação de

[39]http://wiki.hping.org/home

```yaml
- name: 'generic-botnet-pb'
  persitent_volume_path: "/MENTORED_IP_LIST.yaml"
  replicas: 10
  containers:
    - name: 'botnet-pb'
      image: ghcr.io/mentoredtestbed/generic-botnet:latest
      command: ["/entry.sh"]
      args: ["slowloris", "-p", "80"]
      env:
        - name: PROTOCOL
          value: "ICMP"
        - name: TIMEOUT_CMD
          value: "180"
        - name: TIME_WAIT_START
          value: "60"
        - name: ADD_SERVER_IP_TO_COMMAND
      resources:
        requests:
          memory: "64Mi"
          cpu: "100m"
        limits:
          memory: "128M"
          cpu: "200m"
  region: 'ids-pb'
```

Listagem 5.2: YAML descrevendo o atacante *Slowloris*.

endereços IP de origem, dificultando a identificação e mitigação dos ataques. Sua flexibilidade e potência tornam-no uma escolha popular entre os atacantes que buscam realizar ataques DDoS de forma eficaz e difícil de rastrear.

Como ilustrado na Figura 5.14, o ataque será distribuído geograficamente em um ambiente real com três regiões presentes no *Cluster* Nacional da RNP. O nó vítima implementa um servidor *Web* Apache HTTPd desenvolvida com o *micro framework Web flask* padrão localizado em Recife - PE. Os nós atacantes foram divididos em duas regiões, sendo 10 atacantes em João Pessoa - PB e 10 atacantes em Goiânia - GO. Os atacantes foram configurados para utilizar o tipo de pacote TCP SYN com a opção *flood*, o *payload* dos pacotes SYN foram definidos em 1024 *bytes*. Isso significa que uma grande quantidade de pacotes SYN serão enviados para o servidor *Web* vítima, cada um com *payload* de 1024 *bytes* com o objetivo de tornar o serviço *Web* indisponível para os clientes legítimos. Os nós clientes foram posicionados nas mesmas regiões que os atacantes, com 70 clientes em João Pessoa - PB e 70 clientes em Goiânia - GO, realizando requisições GET ao servidor *Web* a cada segundo durante todo o experimento.

A Figura 5.15 representa a vazão de rede e o número de pacotes, obtidos por meio da captura de tráfego com a ferramenta *tshark* na interface de rede do servidor *Web* durante a execução do experimento. Como pode ser observado, a partir do inicio do ataque, há um aumento significativo na vazão de rede em virtude do tráfego malicioso gerado pelo hping3. Durante todo o ataque, o tráfego de rede mantém-se elevado, em linha com o número de atacantes e a capacidade do enlace disponibilizado pelo servidor *Web*. Também é possível perceber na Tabela 5.2 um aumento no tempo de resposta do servidor para os clientes legítimos logo após o início do ataque, afetando a disponibilidade do serviço. O tempo de resposta inicial é de aproximadamente 40 milissegundos, mas durante o ataque, esse tempo sobe para quase 2 segundos. Mesmo após o término do

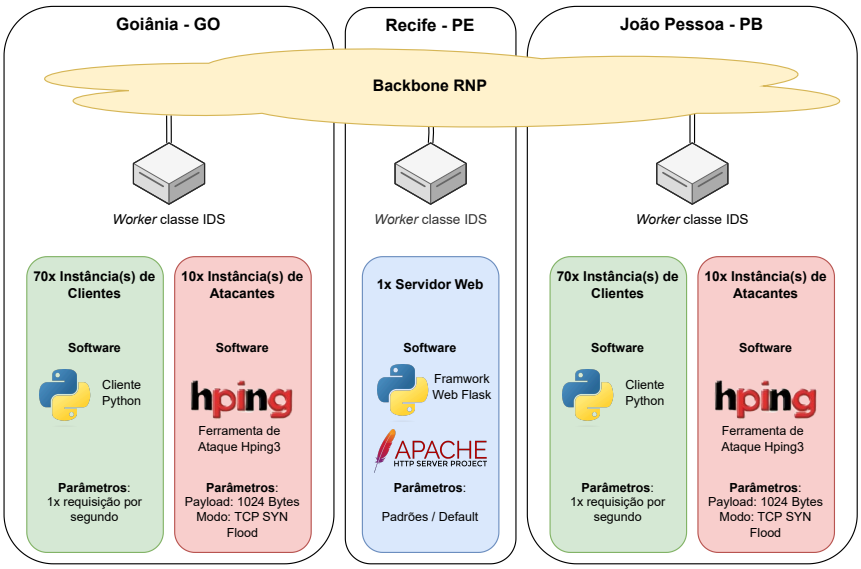


Figura 5.14: Diagrama da disposição dos clientes, atacantes e servidor - Hping.

ataque, o servidor parece enfrentar dificuldades em restabelecer sua disponibilidade pré-ataque. Isso pode ser atribuído ao uso contínuo de recursos alocados durante os ataques, que não são liberados até 60 segundos após o término do ataque.

Tabela 5.2: Resumo das médias do tempo (em segundos) de resposta de clientes na atividade I. As médias foram obtidas dentre todos os clientes para três faixas de tempo: pré ataque (0 à 60 segundos), ataque (60 à 240 segundos) e pós ataque (240 à 300 segundos).

Atividade	Tempo de resposta média dentre clientes (segundos)		
	Pré ataque	**Ataque**	**Pós ataque**
I (hping3)	0.040	1.993	0.463

Em resumo, os ataques de *Flood* DDoS representam uma ameaça significativa à disponibilidade de serviços online e requerem medidas robustas de segurança cibernética. Para mitigar um ataque de *Flood* DDoS, as organizações podem empregar várias estratégias, incluindo o uso de *firewalls*, sistemas de detecção e prevenção de intrusões (IDPS), serviços de mitigação de DDoS baseados em nuvem, balanceamento de carga, e outras técnicas de defesa cibernética. A resposta a um ataque DDoS também pode envolver a cooperação com provedores de serviços de internet (ISPs) e autoridades legais para identificar e desativar os sistemas envolvidos no ataque.

5.4.2. Atividade prática II: negação de serviço com *Slowloris*

O *Slowloris* (YALTIRAKLI, 2015) é uma ferramenta específica para ataques de negação de serviço distribuídos (DDoS) que se destaca pela sua eficácia e simplicidade. O *Slowloris* explora uma vulnerabilidade no protocolo HTTP, aproveitando-se da característica de algumas implementações de servidores *Web* que mantém conexões abertas por longos períodos de tempo.

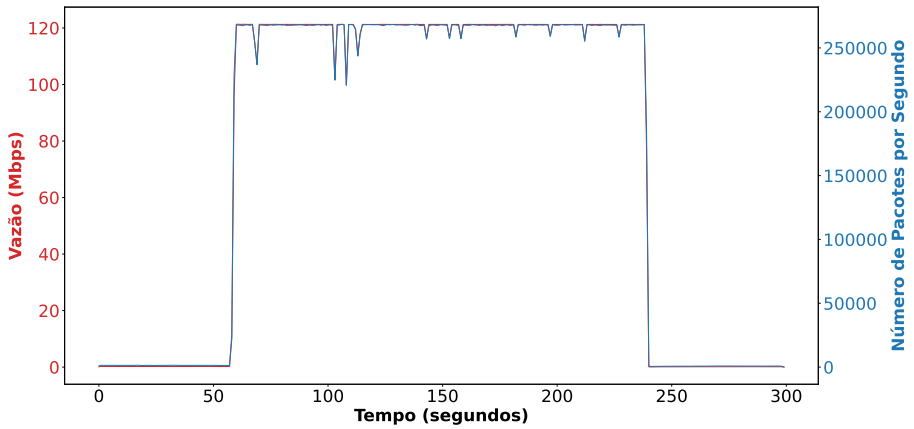


Figura 5.15: Vazão de rede e número de pacotes com o ataque hping.

O modo de operação do *Slowloris* consiste em no envio de solicitações HTTP incompletas, mantendo as conexões abertas com o servidor, mas não enviando a solicitação completa. Isso faz com que o servidor aloque recursos para a conexão aberta, aguardando a conclusão da requisição, enquanto simultaneamente limita o número de conexões que pode atender, eventualmente, levando à sobrecarga e à interrupção do serviço para usuários legítimos. O *Slowloris* é notável por sua capacidade de realizar ataques eficazes com poucos recursos, tornando-o uma escolha popular entre os cibercriminosos. No entanto, trata-se de um ataque amplamente conhecido e sua detecção e mitigação tornaram-se mais comuns à medida que os administradores de sistemas adotam medidas para proteger seus servidores contra esse tipo de ataque. O uso do *Slowloris* em um ambiente controlado é uma forma eficaz de demonstrar os efeitos de um ataque DDoS na camada de aplicação e as estratégias de defesa que podem ser empregadas para mitigá-lo.

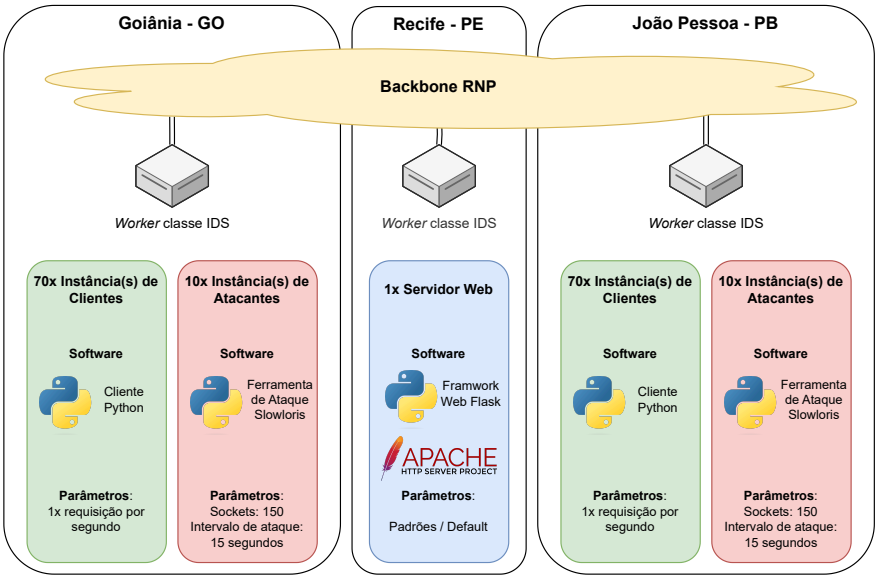


Figura 5.16: Diagrama da disposição dos clientes, atacantes e servidor - *Slowloris*.

Como ilustrado na Figura 5.16 o ataque será distribuído geograficamente em um

ambiente real com três regiões presentes no *Cluster* Nacional - RNP. O nó vítima implementa um servidor *Web* Apache HTTPd desenvolvida com o *micro framework Web flask* padrão localizado em Recife - PE. Os nós atacantes foram divididos em duas regiões, sendo 10 atacantes em João Pessoa - PB e 10 atacantes em Goiânia - GO. Os atacantes executam a implementação `slowloris.py` (YALTIRAKLI, 2015) com paramentos padrão da ferramenta, sendo 150 o número de conexões paralelas abertas simultaneamente em um intervalo de 15 segundos entre o envio de cada cabeçalho na porta 80. Os clientes efetuam requisições GET no intervalo de *1* segundo ao servidor *Web* durante todo o experimento.

A Figura 5.17 representa a vazão de rede e o número de pacotes, esses resultados foram obtidos por meio da captura de tráfego com a ferramente *tshark* na interface de rede do servidor *Web* durante a execução do experimento. Como pode ser observado, existe um pico na vazão de rede, a partir do inicio do ataque, logo resultando na indisponibilidade do serviço. Também é possível observar que diferente do experimento com o hping3, em que durante o período de ataque foi atingido um determinado limite na vazão de rede que se manteve regular e interrupto, com o *Slowloris* ocorreram picos na vazão de rede em intervalos de 15 segundos, mesmo intervalo de tempo que a ferramenta utiliza como padrão para enviar novas requisições. Mesmo esse ataque não ter sobrecarregado a rede, é possível perceber na Tabela 5.3 que ocorreu a indisponibilidade do serviço para os clientes legítimos em virtude do *Slowloris* sobrecarregar o servidor *Web*. Em específico, o *Slowloris* foi capaz de aumentar em média o tempo de resposta do servidor de 40 milissegundos para 8.6 segundos, um ataque muito mais eficiente do que o hping3. Por fim, é possível identificar que mesmo após a conclusão do ataque a maior parte dos clientes legítimos não obtiveram mais resposta em virtude do servidor *Web* ainda estar sobrecarregado com as conexões que foram abertas durante o ataque.

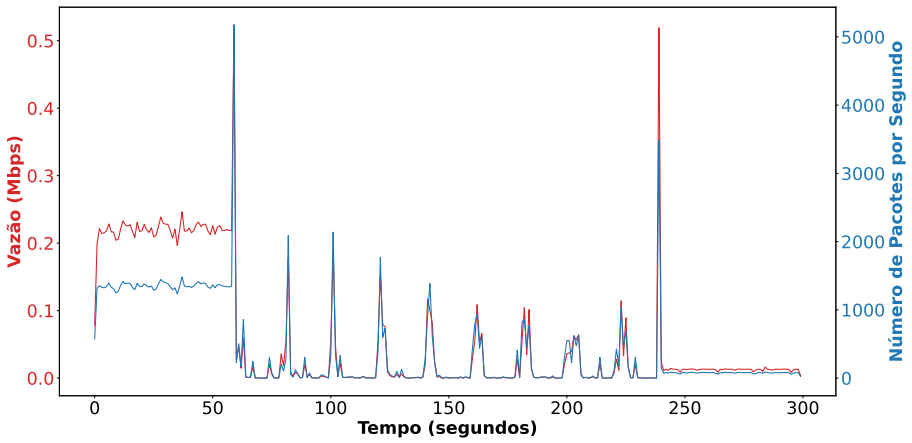


Figura 5.17: Vazão de rede e número de pacotes com o ataque *slowloris*.

Como pôde ser observado, o *Slowloris* é eficaz contra servidores HTTP que mantêm conexões abertas por longos períodos de tempo e não possuem mecanismos eficazes de limitação de conexões por IP ou de detecção de atividades suspeitas. No entanto, essa estratégia tem como base a camada 7 do modelo OSI (aplicação) e muitos servidores e sistemas de proteção contra DoS implementam contramedidas para mitigar esse tipo de ataque, como por exemplo, limitação no número de conexões por IP, fechamento de co-

nexões inativas, e o uso de *firewalls* de aplicativos *Web* (WAFs) que podem detectar e bloquear padrões de tráfego associados ao *Slowloris*. Essa característica não acontece em ataques DoS ou DDoS como os que usam o hping3 e que utilizam a camada 4 (transporte) como base do ataque, tornando complexa a tarefa de diferenciar tráfego de rede legítimo de tráfego de rede malicioso.

Tabela 5.3: Resumo das médias do tempo (em segundos) de resposta de clientes na atividade II. As médias foram obtidas dentre todos os clientes para três faixas de tempo: pré ataque (0 à 60 segundos), ataque (60 à 240 segundos) e pós ataque (240 à 300 segundos).

Atividade	Tempo de resposta média dentre clientes (segundos)		
	Pré ataque	**Ataque**	**Pós ataque**
II (*slowloris*)	0.040	8.618	0.070

5.5. Considerações finais

A crescente complexidade e evolução das ameaças cibernéticas exigem uma abordagem robusta e dinâmica para a pesquisa e desenvolvimento de soluções de segurança. Os *testbeds* desempenham um papel crucial neste contexto, fornecendo ambientes controlados e reprodutíveis que facilitam a experimentação em cibersegurança. Este capítulo destacou diversos *testbeds* especializados, ilustrando suas capacidades únicas, variando desde a simulação de dispositivos IoT até a execução de sofisticados ataques DDoS e técnicas de mitigação.

Por meio dos exemplos detalhados, como o MENTORED *testbed*, DeterLab e FIT IoT-LAB, observa-se a importância dessas infraestruturas no avanço do conhecimento científico e técnico na área. Cada *testbed* apresenta características distintas que atendem a diferentes requisitos de pesquisa, desde a escalabilidade até a fidelidade dos experimentos, refletindo a diversidade de necessidades no campo da cibersegurança.

A implementação de *testbeds* como o MENTORED oferece uma visão sobre como a combinação de tecnologias avançadas e um design estratégico pode resultar em uma plataforma poderosa para testar e validar novas teorias, protocolos de segurança e estratégias de defesa. A colaboração e gestão eficiente de equipes e recursos, aliadas à capacidade de reproduzir cenários realistas, são essenciais para o sucesso dos projetos de pesquisa. Em específico, a implementação do MENTORED *testbed* se destaca no contexto brasileiro devido à sua simplicidade para definir, executar e monitor experimentos por meio de um portal, além da sua infraestrutura baseada no *Cluster* Nacional da RNP.

Como perspectivas futuras para experimentação em cibersegurança, pode-se citar a utilização de tecnologias como aprendizado de máquina e *blockchain*, além de tecnologias emergentes como realidade aumentada e computação quântica, em conjunto com as tecnologias já estabelecidas, bem como a integração com ferramentas modulares relacionadas à segurança. Assim, além dos experimentos atuais de simulação de ataques e defesa, poderá ser ofertado suporte a experimentos mais complexos, como os que envolvem detecção automatizada de ataques, resposta à incidentes automatizada, experimentos de proteção de dados, interoperabilidade segura entre sistemas, privacidade de dados, e gestão de informações sensíveis.

Finalmente, os *testbeds* não só avançam nossa compreensão das ameaças cibernéticas existentes e emergentes, mas também desempenham um papel importante na educação e formação de futuros profissionais de segurança. À medida que enfrentamos desafios cada vez mais sofisticados, a importância desses ambientes experimentais continua a crescer, promovendo um ciclo de inovação contínua que é crucial para manter a segurança e integridade dos sistemas presentes em aplicações reais. Portanto, esses ambientes experimentais são fundamentais para enfrentar os desafios de cibersegurança considerando o atual estado da arte da pesquisa e tecnologias.

Agradecimentos

Este trabalho foi financiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP (#2018/23098-0 e #2023/06265-8), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES (PROSUC), Rede Nacional de Ensino e Pesquisa (RNP) e Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (#141179/2021-0). Os autores utilizaram a ferramenta ChatGPT para revisar a redação de algumas partes deste capítulo.

Referências

ADJIH, Cedric et al. FIT IoT-LAB: A large scale open experimental IoT testbed. In: IEEE. 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT). 2015. P. 459–464.

ALSAEDI, Abdullah et al. TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems. **Ieee Access**, IEEE, v. 8, p. 165130–165150, 2020.

BALDIN, Ilya et al. Fabric: A national-scale programmable experimental network infrastructure. **IEEE Internet Computing**, IEEE, v. 23, n. 6, p. 38–47, 2019.

BENZEL, Terry. The Science of Cyber Security Experimentation: The DETER Project. In: PROCEEDINGS of the 27th Annual Computer Security Applications Conference. Orlando, Florida, USA: Association for Computing Machinery, 2011. (ACSAC '11), p. 137–148. ISBN 9781450306720. DOI: 10.1145/2076732.2076752. Disponível em: <https://doi.org/10.1145/2076732.2076752>.

BERMAN, Mark et al. GENI: A federated testbed for innovative network experiments. **Computer Networks**, Elsevier, v. 61, p. 5–23, 2014.

BUDIGIRI, Gerald et al. Network policies in kubernetes: Performance evaluation and security analysis. In: IEEE. 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit). 2021. P. 407–412.

CHERNYSHEV, Maxim et al. Internet of things (iot): Research, simulators, and testbeds. **IEEE Internet of Things Journal**, IEEE, v. 5, n. 3, p. 1637–1647, 2017.

CHOULIARAS, Nestoras et al. Cyber Ranges and TestBeds for Education, Training, and Research. **Applied Sciences**, v. 11, n. 4, 2021. ISSN 2076-3417. DOI: 10.3390/app11041809. Disponível em: <https://www.mdpi.com/2076-3417/11/4/1809>.

CHUN, Brent et al. Planetlab: an overlay testbed for broad-coverage services. **ACM SIGCOMM Computer Communication Review**, ACM New York, NY, USA, v. 33, n. 3, p. 3–12, 2003.

CLAASSEN, Joris; KONING, Ralph; GROSSO, Paola. Linux containers networking: Performance and scalability of kernel modules. In: IEEE. NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium. 2016. P. 713–717.

CUNHA PONTES, Edgard da et al. FABRIC Testbed from the Eyes of a Network Researcher. In: SBC. ANAIS do II Workshop de Testbeds. 2023. P. 38–49.

DEMEESTER, Piet et al. Fed4fire–the largest federation of testbeds in europe. In: BUILDING the future internet through FIRE. River Publishers, 2022. P. 87–109.

FLANAGAN, Heather et al. Enabling efficient electronic collaboration between LIGO and other astronomy communities using federated identity and COmanage. In: SPIE. SOFTWARE and Cyberinfrastructure for Astronomy II. 2012. v. 8451, p. 530–546.

FORUM, World Economic. **Global Risks Report 2024**. Jan. 2024. https://www.weforum.org/publications/global-risks-report-2024/.

GEMMER, Davi Daniel et al. A Scalable Cyber Security Framework for the Experimentation of DDoS Attacks of Things. In: PROCEEDINGS of IEEE/IFIP Network Operations and Management Symposium (NOMS). Miami, FL, EUA: IEEE/IFIP, mai. 2023.

GOMEZ, Jose et al. A survey on network simulators, emulators, and testbeds used for research and education. **Computer Networks**, v. 237, p. 110054, 2023. ISSN 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2023.110054. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128623004991>.

GOODFELLOW, Ryan; THURLOW, Lincoln; RAVI, Srivatsan. Merge: An Architecture for Interconnected Testbed Ecosystems. **CoRR**, abs/1810.08260, 2018. arXiv: 1810.08260. Disponível em: <http://arxiv.org/abs/1810.08260>.

KORONIOTIS, Nickolaos; MOUSTAFA, Nour; SCHILIRO, Francesco et al. The sair-iiot cyber testbed as a service: A novel cybertwins architecture in iiot-based smart airports. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 24, n. 2, p. 2368–2381, 2021.

KORONIOTIS, Nickolaos; MOUSTAFA, Nour; SITNIKOVA, Elena et al. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. **Future Generation Computer Systems**, Elsevier, v. 100, p. 779–796, 2019.

MEYER, Bruno Henrique; GEMMER, Davi Daniel; SANTANA, Khalil G. Q. de et al. Criação e análise de *datasets* de ataque de negação de serviço usando o Mentored Testbed. In: SBC. ANAIS do XLII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. 2024.

MEYER, Bruno Henrique; GEMMER, Davi Daniel; SCHWARZ, Marcos et al. MENTORED: The Brazilian Cybersecurity Testbed. In: DEMO sessions of IEEE Global Communications Conference (GLOBECOM). Rio de Janeiro, RJ: IEEE, dez. 2022.

MIRKOVIC, Jelena; BENZEL, Terry. Teaching cybersecurity with DeterLab. **IEEE Security & Privacy**, IEEE, v. 10, n. 1, p. 73–76, 2012.

MOUSTAFA, Nour. A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets. **Sustainable Cities and Society**, Elsevier, v. 72, p. 102994, 2021.

NGUYEN, Xuan et al. **Network isolation for Kubernetes hard multi-tenancy**. 2020. Diss. (Mestrado).

PRATES JR, Nelson G et al. Um ambiente de experimentaçao em cibersegurança para internet das coisas. In: SBC. ANAIS do VI Workshop do testbed FIBRE. 2021. P. 68–79.

SÁEZ-DE-CÁMARA, Xabier et al. Gotham testbed: a reproducible IoT testbed for security experiments and dataset generation. **IEEE Transactions on Dependable and Secure Computing**, IEEE, 2023.

SAYFAN, Gigi. **Mastering kubernetes**. Packt Publishing Ltd, 2017.

SCHWAB, Stephen; KLINE, Erik. Cybersecurity experimentation at program scale: Guidelines and principles for future testbeds. In: IEEE. 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). 2019. P. 94–102.

SIATERLIS, Christos; GARCIA, Andres Perez; GENGE, Bela. On the Use of Emulab Testbeds for Scientifically Rigorous Experiments. **IEEE Communications Surveys & Tutorials**, v. 15, n. 2, p. 929–942, 2013. ISSN 1553-877X. DOI: 10.1109/SURV.2012.0601112.00185. Disponível em: <http://ieeexplore.ieee.org/document/6226792/>. Acesso em: 24 abr. 2023.

SIATERLIS, Christos; GENGE, Bela; HOHENADEL, Marc. EPIC: A testbed for scientifically rigorous cyber-physical security experimentation. **IEEE Transactions on Emerging Topics in Computing**, IEEE, v. 1, n. 2, p. 319–330, 2013.

SIBONI, Shachar et al. Security testbed for Internet-of-Things devices. **IEEE transactions on reliability**, IEEE, v. 68, n. 1, p. 23–44, 2018.

THOM, Jay et al. Casting a wide net: An internet of things testbed for cybersecurity education and research. In: IEEE. 2021 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS). 2021. P. 1–8. DOI: 10.23919/SPECTS52716.2021.9639278. DOI: https://doi.org/10.23919/SPECTS52716.2021.9639278.

WROCLAWSKI, John et al. DETERLab and the DETER Project. **The GENI Book**, Springer, p. 35–62, 2016.

YALTIRAKLI, Gokberk. **Low bandwidth DoS tool. Slowloris rewrite in Python**. 2015. https://github.com/gkbrk/slowloris.

Capítulo 6

WebAssembly: Uma Introdução

Tiago Heinrich[1], Beatriz M. Reichert[2], Newton C. Will[3], Rafael R. Obelheiro[2], Carlos A. Maziero[4]

[1]Max Planck Institute for Informatics (MPI)
[2]Universidade do Estado de Santa Catarina (UDESC)
[3]Universidade Tecnológica Federal do Paraná (UTFPR)
[4]Universidade Federal do Paraná (UFPR)

Abstract

In the last decade, Web browsers have become an indispensable resource for users to access the Internet and carry out everyday activities. During this period, different resources were proposed to gain performance, security and practicality in the development of Web applications. One of these resources is WebAssembly, which is a binary and portable format. In this chapter, a complete contextualization of the history of the Web up to its current state will be introduced, with a focus on introducing the WebAssembly format and the impact that this new feature brings to the Web. Discussing performance factors, security, recent trends and open issues.

Resumo

Na última década os navegadores Web se tornaram um recurso indispensável para usuários acessarem a Internet e realizarem atividades cotidianas. Ao decorrer deste período diferentes recursos foram propostos para ganho de desempenho, segurança e praticidade no desenvolvimento de aplicações Web. Um destes recursos é o WebAssembly*, que é um formato binário e portátil. Neste capítulo, será introduzido toda uma contextualização da história da Web até o estado atual, com o foco em introduzir o formato* WebAssembly *e impacto que este novo recurso traz para a Web. Discutindo sobre fatores de performance, segurança, tendências recentes e problemas abertos.*

6.1. Introdução

Inicialmente, a Web era formada por conteúdo exclusivamente estático: páginas HTML com *hiperlinks* e imagens que eram exibidas em um navegador, e arquivos binários que podiam ser salvos para uso local [Fox and Patterson 2021]. Com o decorrer dos anos, as páginas Web passaram a ser dependentes de conteúdos dinâmicos, que são gerados no momento do acesso, muitas vezes de forma personalizada para o usuário. Esses conteúdos dinâmicos podem ser gerados do lado do servidor ou do cliente, mediante a execução de código no contexto do navegador [Stock et al. 2017]. Atualmente, o conteúdo disponível para os usuários ao navegarem pela Web tem o predomínio de conteúdo dinâmico nas páginas que requerem a execução de operações no lado do usuário.

Devido a este "novo" ecossistema, novas linguagens foram introduzidas com o passar do tempo. Historicamente, três linguagens foram propostas com foco no desenvolvimento Web do lado do cliente, sendo elas: Java applets, Flash e JavaScript (JS) [Golsch 2019]. Devido à utilização de diferentes estratégias/*designs*, cada linguagem possui suas peculiaridades, limitações e políticas de segurança.

Java applets e Flash foram descontinuados para aplicativos da Web devido a problemas de desempenho e segurança, respectivamente. JS ganhou então exclusividade em navegadores Web [Feldman 2019]. Dados de dezembro de 2023 estimam que 98,8% dos websites usam JS no lado do cliente [W3Techs 2023].

JS é uma linguagem de fácil uso, que permite o desenvolvimento de aplicativos de pequeno e médio porte. A eficiência e a segurança da linguagem constituem desafios importantes, particularmente em aplicações complexas. Dois problemas encontrados em JS são a falta de tipos e a sobrecarga do interpretador de tempo de execução decorrente da falta de um formato binário. Esses problemas foram identificados em 2015, levando grandes organizações desenvolvedoras a propor o WebAssembly como uma alternativa para superar as limitações encontradas no JS [Bandhakavi et al. 2010, Fraiwan et al. 2012, Yan et al. 2021].

WebAssembly foi lançado em 2017, visando propor um ambiente com maior segurança, velocidade e semântica portátil [Golsch 2019]. Trata-se de um formato de código binário portátil (*bytecode*) projetado para execução segura e eficiente, com uma representação compacta [Rossberg 2024]. Normalmente, ele é usado como um formato de código executável gerado por compiladores para linguagens de alto nível, como C, C++, Go e Rust. Dessa forma, o WebAssembly permite que aplicações e bibliotecas desenvolvidas nessas linguagens sejam executadas em navegadores Web ou mesmo em ambientes nativos [Hoffman 2019].

O *bytecode* WebAssembly geralmente é executado dentro de uma *sandbox* chamada WebAssembly *runtime*. Essa escolha de *design* permite que os aplicativos sejam executados em uma ampla variedade de plataformas, reduz o tamanho das mensagens a serem trocadas com o servidor para recuperar conteúdos, e fornece maior segurança para o ambiente do cliente que executa o conteúdo.

Atualmente, o formato WebAssembly já é suportado por todos os maiores navegadores, como Mozilla Firefox, Google Chrome e Safari [W3C Community Group 2022b]. A gama de aplicações que usam WebAssembly inclui criptografia, processamento de ví-

deo/áudio/gráficos e outras atividades que podem aproveitar as vantagens do *design* do formato. No entanto, o formato não se limita aos navegadores Web. Aplicativos do lado do servidor, serviços distribuídos, Internet of Things (IoT), Cloud e aplicativos móveis também podem aproveitar as vantagens do WebAssembly [Rossberg 2024]. Considerando a rápida adoção do WebAssembly, é importante conhecer as características desta tecnologia e entender seus benefícios e limitações, particularmente no que tange a desempenho e segurança, que são geralmente apontados como suas principais virtudes.

O *design* do WebAssembly conta com diversos mecanismos para proporcionar mais segurança e desempenho para aplicações Web. No entanto, o WebAssembly também pode ser explorado como um vetor de ataques. Este minicurso faz uma introdução à tecnologia WebAssembly, focando nos seguintes objetivos:

Objetivos. Este curso é proposto na modalidade teórica, visando contextualizar o WebAssembly na atualidade, discutindo os principais aspectos do formato, qual o processo de desenvolvimento para o formato, contribuições já propostas para o formato WebAssembly, tendências recentes e problemas de pesquisa ainda em aberto.

Audiência. Este curso está destinado a qualquer público que esteja começando ou que tenha interesse em novas tecnologias encontradas no ecossistema da Web. Espera-se que, ao final do curso, os participantes tenham adquirido um conhecimento sobre WebAssembly que lhes permita compreender os objetivos e o papel desta tecnologia, distinguir seus principais benefícios e limitações, e embasar um maior aprofundamento no assunto.

Estrutura. O restante deste capítulo está organizado como segue. A Seção 6.2 faz um breve histórico do ecossistema Web. A Seção 6.3 discute as principais limitações encontradas no ambiente Web que contribuíram para o desenvolvimento do WebAssembly. O WebAssembly é apresentado na Seção 6.4. A Seção 6.5 aborda aspectos de desempenho do WebAssembly. A Seção 6.6 traz uma revisão abrangente de aspectos de segurança da tecnologia. A Seção 6.7 examina estratégias de mitigação, monitoramento e detecção de ameaças em WebAssembly. A Seção 6.8 discute tendências recentes e problemas abertos de pesquisa envolvendo WebAssembly. Por fim, a Seção 6.9 apresenta as considerações finais do capítulo.

6.2. História da Web

No início, a Web consistia em páginas de conteúdo estático, i.e., páginas HyperText Markup Language (HTML) com texto, imagens e *hiperlinks* que permitiam acesso a outras páginas ou arquivos (tipicamente arquivos binários que podiam ser salvos localmente pelo cliente, sem serem exibidos pelo navegador) [Fox and Patterson 2021]. A interação do usuário com as páginas resumia-se a acessar os *hiperlinks* disponíveis.

Com o decorrer dos anos, as páginas passaram a ser dependentes de conteúdos dinâmicos. A primeira fase consistia na execução de código do lado do servidor para gerar, *on-the-fly*, páginas HTML que implementavam uma interface com o usuário. O próximo passo na evolução foi o surgimento da tecnologia Asynchronous JavaScript and XML (AJAX). Com AJAX, as páginas passaram a incluir código JS que é executado no lado do cliente (no âmbito do navegador) para definir ou alterar o comportamento da página exibida [Fox and Patterson 2021]. Atualmente, observa-se que o conteúdo dinâmico

presente nas páginas é predominantemente gerado a partir de operações no lado do cliente [Stock et al. 2017].

Devido ao aumento na complexidade das aplicações Web, nota-se a exigência de um formato de entrega que seja portátil, compacto, rápido de executar e seguro. Tentativas anteriores de resolver estes problemas incluem JS, Java applets, Flash, ActiveX, Native Client (NaCl) e asm.js [Golsch 2019].

O JavaScript (JS) é a única linguagem de programação com suporte nativo na Web [Haas et al. 2017]. Atualmente é o principal componente das aplicações Web do lado do cliente. O código fonte é transmitido em texto simples e interpretado por um suporte de execução, o qual é executado isolado em uma *sandbox* e se comunica com o navegador da Web [Golsch 2019]. Entretanto, o JS é lento para a maioria das aplicações complexas, e se tornou um alvo de compilação para outras linguagens.

Java applets permitem a execução de código Java em navegadores da Web. Esse código é executado em uma Java Virtual Machine (JVM) isolada, a qual é integrada ao navegador ou pode ser instalada com um *plugin* [Golsch 2019]. Devido às vulnerabilidades de segurança, muitos navegadores não têm mais suporte aos Java applets.

O Flash possui uma abordagem semelhante ao JS, mas requeria a instalação de um *plugin* adicional para a execução [Golsch 2019]. O uso do Flash foi desaconselhado devido a problemas de segurança. Em dezembro de 2020 a Adobe anunciou o encerramento do suporte e da distribuição do Flash [Adobe 2021].

Em 1996, a Microsoft apresentou a abordagem ActiveX para assinatura de código de binários para execução na Web [Microsoft 1996]. Os objetos ActiveX incluem código de máquina compilado e, portanto, não são portáteis. Do ponto de vista de segurança, ActiveX é vulnerável pois a execução não é isolada do sistema operacional, como em uma *sandbox* [Golsch 2019].

Em 2009, a Google introduziu uma tecnologia chamada NaCl, que permite que código nativo seja executado em uma *sandbox* [Yee et al. 2010]. O código fonte pode ser compilado em um módulo, o qual é executado no ambiente interno do navegador Chromium. Como o NaCl é um subconjunto do código de máquina de uma arquitetura específica, ele não é portátil [Haas et al. 2017]. Posteriormente o Google introduziu o Portable Native Client (PNaCl), o qual é uma extensão do NaCl e permite a execução do código nativo independente de plataforma [Donovan et al. 2010]. Assim como o NaCl, o PNaCl é suportado apenas pelos navegadores Google Chrome e Chromium.

Em 2013, a Mozilla apresentou o asm.js [Herman et al. 2014], uma linguagem intermediária que, a partir do código nativo, gera sistematicamente um código usando um transcompilador (um compilador especializado em converter o código-fonte de um programa em outra linguagem fonte). Por exemplo, o Emscripten [Zakai 2011] pode ser usado para converter C/C++ em código JS e WebAssembly.

Após as experiências com Java, ActiveX, Flash, NaCl e asm.js, e considerando as limitações do JS, foi proposto o WebAssembly. Este consiste em um novo formato portátil, eficiente em tamanho e tempo de carregamento, adequado para compilação na Web. Em março de 2017 a Minimum Viable Product (MVP) foi concluída para o WebAssem-

bly, com o Safari já adicionando o suporte no final do mesmo ano. Em 2018 os primeiros *drafts* de especificações para a JS *Interface* e Web API foram apresentadas. O primeiro *draft* do WebAssembly 2.0 foi anunciado em 2022 (com a última atualização em março de 2024) [Rossberg 2024].

6.3. Limitações da Web

Os navegadores Web evoluíram significativamente nos últimos anos. Atualmente os navegadores são executados em diferentes tipos de *hardware*, como celulares, tablets e computadores [Grosskurth and Godfrey 2006]. No entanto, a maioria dos navegadores Web ainda usam uma arquitetura monolítica introduzida em 1993 pelo NCSA Mosaic [Barth et al. 2008, Rokicki 2022]. Do ponto de vista da segurança, os navegadores monolíticos são executados em um único domínio de proteção. Dessa forma, um invasor capaz de explorar uma vulnerabilidade não corrigida pode comprometer toda a instância do navegador. Além disso, um invasor pode executar código arbitrário na máquina do usuário com os privilégios deste.

Antes do WebAssembly houve outras tecnologias com foco no desenvolvimento Web do lado do cliente (*e.g.*, Flash, ActiveX, Java applets, asm.js, e JS). Entretanto, cada uma tem suas peculiaridades, limitações e políticas de segurança. O Flash, por exemplo, exigia que o usuário instalasse o Flash Player [Golsch 2019]. Além de apresentar carregamento lento e gerar arquivos grandes. Do ponto de vista da segurança, o Flash possuía vulnerabilidades no código, as quais permitiam que invasores infiltrassem código malicioso no computador dos usuários.

O ActiveX apresentado pela Microsoft não é executado em uma *sandbox*. Devido a isso, quando um controle ActiveX é instalado no computador do usuário, este torna-se parte do sistema operacional, e é capaz de adulterar a máquina [Golsch 2019]. Já o grande problema dos Java applets é que estes são executados automaticamente no navegador, sem o consentimento do usuário, tornando os recursos de segurança do Java insuficientes.

O JS é a linguagem mais utilizada por aplicações Web ao considerar a execução de conteúdo no lado do cliente [Yan et al. 2021]. Porém, o desempenho do JS é frequentemente considerado uma grande limitação na prática. Os programas JS são distribuídos em código fonte que precisa ser analisados e interpretados em tempo de execução. Os programas WebAssembly, por sua vez, são entregues como binários compilados, os quais podem ser carregados e executados mais rapidamente. Além disso, os programas WebAssembly geralmente são criados usando compiladores, os quais são responsáveis por compilar programas existentes em linguagens de programação de alto nível como C/C++ para o bytecode WebAssembly [Yan et al. 2021].

Das tecnologias predecessoras do WebAssembly é possível observar o uso de linguagens de alto nível como C/C++ na Web apenas no asm.js. Entretanto, com a criação do WebAssembly, o asm.js se tornou obsoleto, pois o WebAssembly possui um formato de *bytecode* que é mais rápido de analisar.

6.4. WebAssembly

O WebAssembly, também conhecido como Wasm, é um formato projetado para ser um alvo binário para linguagens de programação. A portabilidade permite aos desenvolvedores escolher a linguagem mais adequada para implementação de uma aplicação. Portanto, uma variedade de linguagens de alto nível não suportadas anteriormente no ambiente Web podem agora ser usadas, facilitando a tarefa dos desenvolvedores de software. Com linguagens de alto nível, os desenvolvedores têm mais flexibilidade, acesso a estruturas e um alvo binário compacto.

O *bytecode* WebAssembly é executado dentro de uma máquina virtual baseada em pilha. As aplicações podem ser executadas em um suporte de tempo de execução nativo (uma ferramenta de linha de comando) no lado do servidor, dentro do navegador Web ou até mesmo em dispositivos IoT. Da perspectiva do lado do cliente (como ilustrado na Figura 6.1), os binários compactos são recuperados de um servidor para serem executados em um ambiente *sandbox*, oferecendo um ganho de desempenho em relação a linguagens interpretadas como JS.

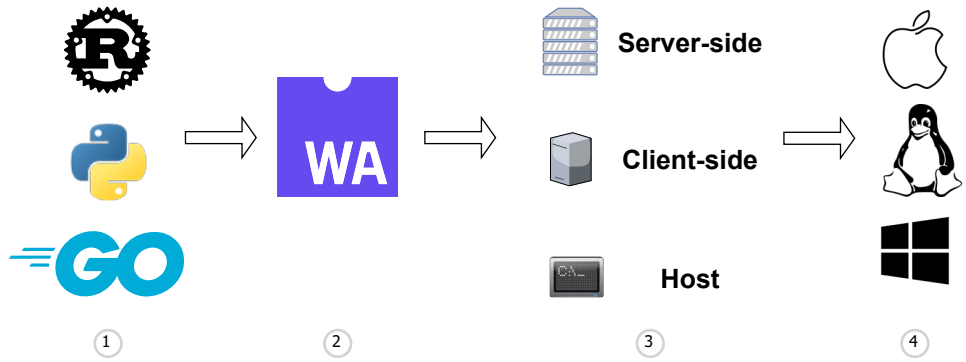


Figura 6.1. Processo de desenvolvimento em WebAssembly. (1) Um conjunto variado de linguagens pode ser utilizado para o desenvolvimento de aplicações (2) Compiladores específicos são utilizados para gerar os binários; (3) Estes binários podem ser executados em um conjunto variado de ambientes, inclusive (4) diferentes plataformas.

A Figura 6.2 apresenta a interação esperada para uma aplicação WebAssembly. Um binário WebAssembly é executado dentro de um ambiente sandbox, onde as instruções são adicionadas e retiradas da pilha, as informações são armazenadas na memória linear e recursos externos podem interagir com o ambiente *sandbox* por meio de uma função exposta [Rourke 2018]. Um exemplo de interação externa entre código Wasm e JS por meio de funções expostas é apresentado na Figura 6.2.

As instruções em uma máquina baseada em pilha assumirão que a maioria dos operandos está na pilha, em vez de nos registradores [Hoffman 2019]. A pilha WebAssembly é uma estrutura Last-In, First-Out (LIFO), qualquer interação e comandos com a pilha dentro do ambiente virtual serão limitados ao elemento no topo da pilha [Battagline 2021]. Essas opções de *design* permitem fácil portabilidade de outras linguagens e binários com tamanho pequeno [Hoffman 2019].

O formato WebAssembly suporta quatro tipos de dados: `i32` (número inteiro de

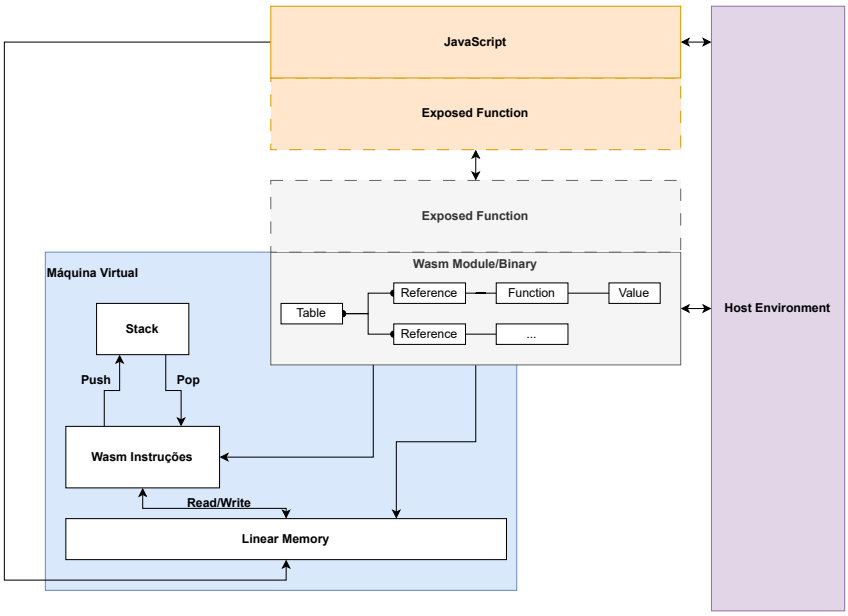


Figura 6.2. Exemplo de interação entre um módulo WebAssembly e recursos expostos.

32 bits), `i64` (número inteiro de 64 bits), `f32` (números de ponto flutuante de 32 bits) e `f64` (números de ponto flutuante de 64 bits). A sinalização intrínseca à representação numérica só é realizada quando uma operação é realizada, com operações específicas para dados com e sem sinal [Hoffman 2019]. *Strings* só podem ser usadas por meio de memória linear. O controle de *strings* e outros elementos encontrados na memória linear para recursos externos como JS é feito passando a posição e o comprimento desses elementos na memória [Battagline 2021]. Como o WebAssembly não possui *heap*, não há suporte a programação orientada a objetos. A memória linear é um bloco de bytes gerenciado pelo código Wasm; caso seja necessário mais espaço, os blocos podem ser incrementados em páginas, até um limite predefinido. O acesso à memória do *host* através da memória linear não é possível (exceto nos casos que seja definido explicitamente o acesso externo ao recurso).

A Figura 6.3 apresenta a estrutura de um binário WebAssembly. Ele é composto por um módulo WebAssembly, e estruturado em três seções: *Preâmbulo*, *Standard* e *Custom*. O *Preâmbulo* indica que o arquivo é um módulo WebAssembly e o formato da versão que está sendo usada. A seção *Standard* apresenta seções conhecidas que possuem funcionalidades específicas. A Tabela 6.3 mostra onze seções padrão. Nenhuma seção é exigida por um módulo Wasm, aparecendo apenas quando necessário. A seção *Custom* pode estar em qualquer posição de um módulo, permitindo a inclusão de dados que podem ser usados para depuração ou funções [Hoffman 2019].

6.4.1. WASI

A Application Programming Interface (API) padrão definida para desenvolvimento WebAssembly é conhecida como WebAssembly System Interface (WASI). Ela define as regras para o suporte de execução e as interações que o WebAssembly realiza com o ambiente. Através do WASI, o WebAssembly pode realizar operações nativas encontradas

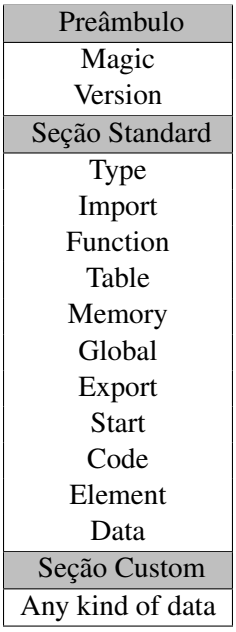


Figura 6.3. Estrutura de um binário WebAssembly (módulo WebAssembly).

em outras linguagens de programação, além de garantir os níveis de segurança definidos para o WebAssembly [Battagline 2021].

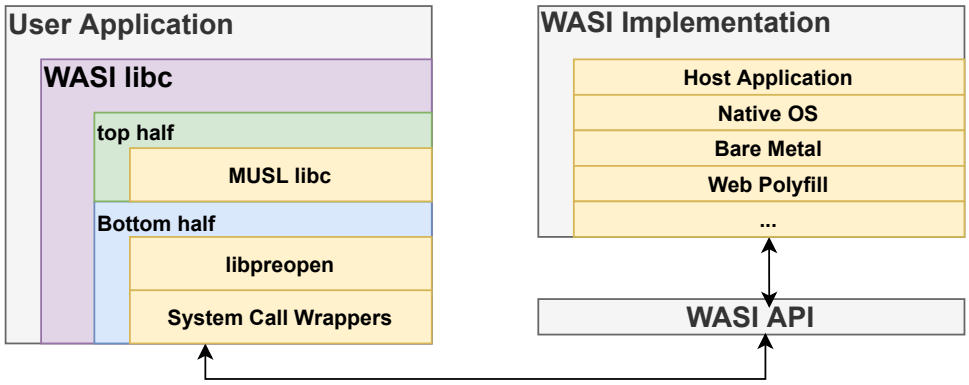


Figura 6.4. Arquitetura da WebAssembly System Interface (WASI) [BytecodeAlliance 2021a].

O foco inicial da WASI API é fornecer acesso a recursos como arquivos e rede. A WASI API usa WASI libc (biblioteca C padrão), que pretende ser uma interface libc. Esta implementação pode ser dividida em *bottom half*, que é composta por `libpreopen` e *wrappers* de chamadas de sistema; e *top half*, que é uma implementação da MUSL libc[1] [BytecodeAlliance 2021a]. A Figura 6.4 mostra esta arquitetura, com o fluxo de comunicação esperado. O *wrapper* de chamadas de sistema é responsável por fazer chamadas para a API WASI. Consequentemente, a API WASI faz as chamadas correspondentes à implementação WASI presente no sistema. Essas chamadas podem ser processadas pelo sistema operacional ou pelo suporte de tempo de execução de alguma linguagem.

[1]MUSL libc implementa a biblioteca padrão C sobre as chamadas do sistema Linux [Felker 2023].

Toda interação entre a aplicação e o sistema operacional é realizada através de chamadas WASI. Essas chamadas possuem funcionalidade semelhante às chamadas de sistema, pois sem elas um aplicativo não seria capaz de acessar os recursos do sistema.

6.4.2. Desenvolvimento e uso

O desenvolvimento de aplicações no formato WebAssembly pode ser realizado de duas formas: (i) o porte de aplicações desenvolvidas em outras linguagens como Rust, C/C++ e Go; ou (ii) através de uma representação legível chamada WebAssembly Text (WAT), que permite aos desenvolvedores escrever código WASM usando Expressões S ou estilos de lista de instruções lineares [Battagline 2021, W3C Community Group 2022b].

Emscripten [Emscripten 2021] e Wasmtime [BytecodeAlliance 2021b] são os dois compiladores mais populares para WebAssembly. Ambos seguem o padrão WASI mas têm objetivos diferentes. Emscripten é baseado na infraestrutura do compilador LLVM [LLVM 2023] e está focado na compilação de código C/C++ em WebAssembly. O principal alvo de execução são os navegadores da web, e os arquivos JS e HTML necessários são empacotados junto com o código *.wasm* gerado.

Wasmtime gera aplicativos *.wasm* que podem ser executados no navegador ou como um aplicativo no *host* sem a necessidade de um mecanismo de navegador. Devido a essas características, o Wasmtime fornece uma gama de ferramentas de depuração para avaliar a interação do código com o host e também fornece uma maneira de recuperar chamadas WASI do tempo de execução de uma aplicação.

Um total de 45 chamadas já estão implementadas no Wasmtime; estas são as chamadas que seguem o padrão WASI (vamos nos referir a essas chamadas por *chamadas WASI*). Wasmtime segue os *namespaces* (*snapshots*) definidos para a WASI API para controlar o suporte de recursos do compilador e, consequentemente, também controla o tipo de chamadas WASI suportadas em uma versão específica. A versão atual 19.0.2 do Wasmtime define as chamadas WASI no *snapshot_preview1*, e futuras versões de *snapshot* permitirão que os desenvolvedores usem funções de um *snapshot* diferente, o que consequentemente fornecerá mais chamadas.

Em geral, os compiladores para WebAssembly estão em estágios iniciais de desenvolvimento e as mudanças são frequentes. A vantagem do compilador Wasmtime está na forma como as chamadas WASI são tratadas, com a possibilidade de executar e observar essas aplicações sem a necessidade de utilizar um navegador.

WebAssembly possui uma comunidade ativa que está constantemente implementando melhorias e novos recursos. A versão 1.0 do WebAssembly não suportava um coletor de lixo para gerenciamento de memória até o final de 2023 e não fornece acesso direto ao Document Object Model (DOM) [Rourke 2018].

Uma aplicação WebAssembly (ou um arquivo WAT) contém um módulo, que será composto por funções e variáveis (seguindo a estrutura apresentada na Tabela 6.3). As operações de exportação e importação fornecem acesso às funções do WebAssembly. A operação de exportação disponibiliza uma função Wasm para outras operações no ambiente (por exemplo, um valor compartilhado JS durante o processo de execução). As funções no ambiente também podem ser importadas para o WebAssembly usando a ope-

ração de importação.

Visando a melhor compreensão do formato, as Listagens 6.1 e 6.2 apresentam uma implementação de uma função *Hello, world!* em Rust e WebAssembly (usando o formato WAT) respectivamente. A implementação em Rust consiste de uma função *main* que usa a operação *println* para imprimir a mensagem na tela do usuário (linhas 1–3). O código é compilado usando *rustc* (linha 5), e o binário gerado é executado na linha de comando (linhas 6 e 7).

```
1 fn main() {
2    println!("Hello, world!");
3 }
4
5 $ rustc main.rs
6 $ ./main
7 Hello, world!
```

Listagem 6.1. Exemplo de uma implementação de *Hello, world!* em Rust.

A versão WebAssembly é diferente devido à limitação de tipos do formato. Tipos como *strings*, objetos ou dicionários, que podem ser encontrados em muitas linguagens de programação de alto nível, não são suportadas pelo formato WebAssembly [Sletten 2022]. Desta forma, vamos utilizar este exemplo para explanar o formato e as principais operações efetuadas durante o processo de execução.

O formato WAT adota uma representação similar à de linguagens como Lisp. Linhas iniciadas por `;;` são comentários. O início de uma aplicação é definido pela palavra reservada `module` que define o início de uma aplicação, como toda a estrutura que será utilizada pelo sistema para carregar os recursos necessários para executar a aplicação, seguindo os campos estruturais apresentados na Figura 6.3. O código do exemplo (Listagem 6.2) está organizado internamente em sete partes:

(1) A aplicação WebAssembly pode precisar de acesso a recursos externos para realizar suas operações corretamente. A palavra chave `import` realiza a importação desses recursos. Neste caso, o módulo requer a função `fd_write()`, encontrada no *namespace* `wasi_unstable`. Essa função permite escrever um ou mais vetores de bytes em um arquivo, e é usada no exemplo para enviar uma *string* para a saída padrão (*stdout*). `fd_write()` recebe quatro parâmetros: (i) o descritor de arquivo; (ii) início do primeiro vetor na memória linear (deslocamento em bytes); (iii) número de elementos do vetor; (iv) o deslocamento em bytes na memória linear onde a função retorna o número de bytes escritos.

(2) Como esta informação precisa ser escrita em algum lugar, declaramos a memória linear (linha 6). Uma função dentro do módulo só se torna visível no momento que ela é exportada (`export`), desta forma exportamos a memória, já que a função `fd_write()` que foi importada vai precisar interagir com a memória.

(3) Agora que temos acesso à memória e à função de escrita, utilizamos o elemento `data` para escrever a *string Hello, world!* na memória. Como o segundo argumento de `fd_write()` são vetores de bytes, a memória fica organizada da seguinte forma:

- bytes 0–3: deslocamento da *string* (em relação ao início do vetor, isto é, ao byte 0);
- bytes 4–7: comprimento da *string* (em bytes);
- bytes 8–19: a *string* propriamente dita; e
- bytes 20–23: espaço que será usado para armazenar o número de bytes escritos por `fd_write()`.

Logo, a *string* é armazenada em um deslocamento de 8 bytes em relação ao início da memória linear.

(4) Para juntar todas estas operações/funções, exportamos uma nova função `main`.

(5) A primeira operação realizada pela função `main` consiste em criar o vetor correspondente à *string*. Declaramos um ponteiro para o início da *string* (linha 15), considerando o nosso deslocamento de 8 bytes, e que a *string* possui tamanho de 12 bytes (linha 16).

(6) Realizamos a chamada de `fd_write()`. O descritor de arquivo é 1, que corresponde à saída padrão (linha 20). O vetor a ser escrito está na posição 0 da memória linear (linha 21). O número de elementos desse vetor é 1 (linha 22). O número de bytes escritos por `fd_write()` será armazenado na posição 20 da memória linear (linha 23).

(7) Descartamos os *bytes* escritos no topo da pilha.

Resultado: Por fim, podemos observar a execução do módulo WebAssembly utilizando o *wasmtime*. Neste caso por utilizarmos o *wasmtime* não é necessário compilar e executar, já que o compilador realiza todas estas operações automaticamente.

```
1  (module
2     ;; (1)
3     (import "wasi_unstable" "fd_write" (func $fd_write (param i32 i32 i32 i32) (
          result i32)))
4
5     ;; (2)
6     (memory 1)
7     (export "memory" (memory 0))
8
9     ;; (3)
10    (data (i32.const 8) "Hello, world!\n")
11
12    ;; (4)
13    (func $main (export "_start")
14       ;; (5)
15       (i32.store (i32.const 0) (i32.const 8))
16       (i32.store (i32.const 4) (i32.const 12))
17
18       ;; (6)
19       (call $fd_write
20          (i32.const 1)
21          (i32.const 0)
22          (i32.const 1)
23          (i32.const 20)
24       )
25       ;; (7)
```

```
      drop
   )
)

$ curl https://wasmtime.dev/install.sh -sSf | bash
$ wasmtime main.wat
Hello, world!
```

Listagem 6.2. Exemplo de *Hello, world!* em WebAssembly [BytecodeAlliance 2024].

O formato WebAssembly pode ser utilizado por outras linguagens para a execução de funcionalidades específicas. Por exemplo, uma aplicação implementada em Rust pode invocar uma função em WebAssembly. Para explanar este caso, apresentamos a Listagem 6.3, que é um código Rust que imprime o máximo divisor comum (mdc) de dois números (6 e 27); o cálculo em si é efetuado por uma função `gcd()`, implementada em WebAssembly. O código Rust envolve os seguintes passos:

(1) Importamos a *crate* necessária para realizar estas operações.

(2) Para que a aplicação em Rust consiga acessar os recursos da função exposta no módulo WebAssembly, precisamos conseguir invocar o módulo (linha 8). Para alcançar este ponto, precisamos realizar a análise sintática e armazenar este módulo (linha 7).

(3) Uma referência para a função `gcd` do módulo WebAssembly exposto é armazenada em `gcd`.

(4) Invocamos a função usando `gcd.call()` e imprimimos o resultado.

```rust
// (1)
use wasmtime::*;

fn main() -> Result<()> {
   // (2)
   let mut store = Store::<()>::default();
   let module = Module::from_file(store.engine(), "examples/gcd.wat")?;
   let instance = Instance::new(&mut store, &module, &[])?;

   // (3)
   let gcd = instance.get_typed_func::<(i32, i32), i32>(&mut store, "gcd")?;

   // (4)
   println!("gcd(6, 27) = {}", gcd.call(&mut store, (6, 27))?);
   Ok(())
}
```

Listagem 6.3. Código Rust que usa uma função Wasm para calcular o máximo divisor comum (mdc) [BytecodeAlliance 2024].

A função WebAssembly (Listagem 6.4) calcula o mdc usando o algoritmo de Euclides. O código (linha 2) define a função `gcd`, que recebe dois parâmetros inteiros e retorna o mdc (também um inteiro). Esta função é exposta (linha 26) para permitir as interações com o ambiente externo.

```
(module
  (func $gcd (param i32 i32) (result i32)
```

```
3     (local i32)
4     block ;; label = @1
5       block ;; label = @2
6         local.get 0
7         br_if 0 (;@2;)
8         local.get 1
9         local.set 2
10        br 1 (;@1;)
11      end
12      loop ;; label = @2
13        local.get 1
14        local.get 0
15        local.tee 2
16        i32.rem_u
17        local.set 0
18        local.get 2
19        local.set 1
20        local.get 0
21        br_if 0 (;@2;)
22      end
23    end
24    local.get 2
25  )
26  (export "gcd" (func $gcd))
27 )
```

Listagem 6.4. Função GCD em WebAssembly [BytecodeAlliance 2024].

Agora vamos considerar como o formato WebAssembly seria utilizado junto com o JS. Para realizar esta tarefa, precisamos de uma implementação de *Hello, world!* em WebAssembly (Listagem 6.5) que será compilada para gerar um arquivo *.wasm*; um aplicação em JS que será responsável por preparar o ambiente e invocar a aplicação em WebAssembly (Listagem 6.6); e um arquivo *.html* para definir um ambiente mínimo (Listagem 6.7) [Apodaca 2020].

O código do arquivo *hello.wat* pode ser observado na Listagem 6.5. As linhas 3 e 4 importam as dependências criadas no lado do JS. Em seguida a linha 7 escreve o texto *"Hello, World!"*, começando no índice de memória 0 usando o operador de dados. Por fim, é exportada uma função chamada `hello` na linha 10. Dentro desta função, dois valores são inseridos sequencialmente (0 e 13). O primeiro é um deslocamento para a memória compartilhada definida no lado do JS. O segundo é o tamanho da *string "Hello, World"*. O operador `call` retira os dois valores da pilha e invoca a função `log()` com esses dois parâmetros.

```
1  (module
2   ;; (1)
3   (import "env" "memory" (memory 1))
4   (import "env" "log" (func $log (param i32 i32)))
5
6   ;; (2)
7   (data (i32.const 0) "Hello, World!")
8
9   ;; (3)
10  (func (export "hello")
11    ;; (4)
12    i32.const 0
13    i32.const 13
14    call $log
15  )
16 )
```

Listagem 6.5. Exemplo de *Hello, world!* [Apodaca 2020].

A Listagem 6.6 apresenta como esta interação vai ocorrer no lado do JS. Este processo segue a lógica apresentada na Listagem 6.3, onde a aplicação em JS precisa invocar o binário WebAssembly, neste caso precisa ser invocada a função `log()`. Esta função vai acessar a memória compartilhada, que foi exposta no módulo WebAssembly, para ler a *string* de acordo com os parâmetros de posição e tamanho.

```
const memory = new WebAssembly.Memory({ initial: 1 });

const log = (offset, length) => {
  const bytes = new Uint8Array(memory.buffer, offset, length);
  const string = new TextDecoder('utf8').decode(bytes);

  console.log(string);
};

(async () => {
  const response = await fetch('./hello.wasm');
  const bytes = await response.arrayBuffer();
  const { instance } = await WebAssembly.instantiate(bytes, {
    env: { log, memory }
  });

  instance.exports.hello();
})();
```

Listagem 6.6. Função em JS para invocar o binário WebAssembly [Apodaca 2020].

Na Listagem 6.7 apresentamos o HTML que será encontrado no servidor Web, e que chama o *script* em JS para ser executado.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Hello, World! in WebAssembly</title>
  </head>
  <body>
    <script src="hello.js" type="module"></script>
  </body>
</html>
```

Listagem 6.7. HTML da página [Apodaca 2020].

O último exemplo das funcionalidades do WebAssembly consiste em portar aplicações de uma linguagem específica para Wasm. A Listagem 6.8 apresenta uma função em Rust que será portada para WebAssembly. Esta função recebe dois valores `usize` e realiza a soma dos dois valores, retornando um `usize`. Usamos `no_mangle` para que os nomes de funções sejam mantidos para posterior análise, e `extern` para que o compilador trate a função como estejam utilizando a *C ABI*. O alvo de compilação `wasm32-unknown-unknown` é usado para uma função que será executada dentro de um ambiente JS, enquanto o alvo `wasm32-wasi` é usado para código executado no seu próprio *runtime*. Desta forma, o resultado deste processo de compilação consiste em um binário que pode ser executado, por exemplo, em um ambiente Web similar ao da Listagem 6.5. Caso seja desejado executar o código em Rust apresentado na Listagem 6.1 com o *wasmtime*, só é necessário realizar a troca do alvo para `wasm32-wasi` como apresentado na Listagem 6.9.

```
#[no_mangle]
pub extern "C" fn add(value_a: usize, value_b: usize) -> usize {
    value_a + value_b
}

$ rustup target add wasm32-unknown-unknown
$ cargo build --target=wasm32-unknown-unknown
$ ls target/wasm32-unknown-unknown/debug/add_wasm.wasm
```

Listagem 6.8. Exemplo de função em Rust que será compilada para WebAssembly [Levick 2024].

```
$ rustup target add wasm32-wasi
$ rustc hello.rs --target=wasm32-wasi
$ wasmtime hello.wasm
Hello, world!
```

Listagem 6.9. Execução da função Rust usando *wasmtime* [Levick 2024].

Agora podemos entender melhor o conteúdo encontrado no binário (o nosso módulo WebAssembly que foi gerado a partir de um código em Rust). Vamos utilizar o *wasm-tools*[2] para inspecionar os binários. Ao executar um *wasm-tools print –skeleton target/wasm32unknownunknown/debug/add_wasm.wasm* conseguimos entender a estrutura encontrada no módulo WebAssembly, como apresentado na Listagem 6.10. Ao comparar estruturas anteriores, encontramos o `export` da memória linear (linha 29), bem como o da função `add` (linha 30) criada na Listagem 6.8 e a definição da nossa função `add` (linha 15), que recebe dois parâmetros.

```
(module
  (type (;0;) (func (param i32 i32)))
  (type (;1;) (func (param i32 i32 i32) (result i32)))
  (type (;2;) (func (param i32 i32) (result i32)))
  (type (;3;) (func (param i32 i32 i32)))
  (type (;4;) (func (param i32 i32 i32 i32) (result i32)))
  (type (;5;) (func (param i32)))
  (type (;6;) (func (param i32 i32 i32 i32)))
  (type (;7;) (func (param i32) (result i32)))
  (type (;8;) (func (param i32 i32 i32 i32 i32 i32)))
  (type (;9;) (func))
  (type (;10;) (func (param i32 i32 i32 i32 i32 i32) (result i32)))
  (type (;11;) (func (param i32 i32 i32 i32 i32) (result i32)))
  (type (;12;) (func (param i64 i32 i32) (result i32)))
  (func $add (;0;) (type 2) (param i32 i32) (result i32) ...)
  (func $__rust_alloc (;1;) (type 2) (param i32 i32) (result i32) ...)
  (func $__rust_dealloc (;2;) (type 3) (param i32 i32 i32) ...)
  (func $__rust_realloc (;3;) (type 4) (param i32 i32 i32 i32) (result i32) ...)
  (func $__rust_alloc_error_handler (;4;) (type 0) (param i32 i32) ...)
  (func $_ZN36_$LT$T$u20$as$u20$core..any..Any$GT$7type_id17h2a84f20e718440dfE
      (;5;) (type 0) (param i32 i32) ...)
...
  (func $_ZN17compiler_builtins3mem6memcpy17h7037a3a0dead1e85E (;54;) (type 1) (
      param i32 i32 i32) (result i32) ...)
  (func $memcpy (;55;) (type 1) (param i32 i32 i32) (result i32) ...)
  (table (;0;) 20 20 funcref)
  (memory (;0;) 17)
  (global $__stack_pointer (;0;) (mut i32) i32.const 1048576)
  (global (;1;) i32 i32.const 1049701)
  (global (;2;) i32 i32.const 1049712)
  (export "memory" (memory 0))
  (export "add" (func $add))
```

[2]Wasm-tools pode ser instalado com *cargo install wasm-tools*, instruções em `https://github.com/bytecodealliance/wasm-tools`.

```
31  (export "__data_end" (global 1))
32  (export "__heap_base" (global 2))
33  (elem (;0;) (i32.const 1) ...)
34  (data $.rodata (;0;) (i32.const 1048576) ...)
35  (@custom ".debug_abbrev"
36  ...
```

Listagem 6.10. Exemplo da estrutura encontrada no WebAssembly compilado do Rust [Levick 2024] (parte do *output* foi omitida).

Na prática, o WebAssembly é usado para criptografia, vídeo, áudio, gráficos e outras atividades que possam aproveitar as vantagens do *design* do formato. No entanto, o formato WebAssembly não se limita ao navegador da web. Aplicativos do lado do servidor, serviços distribuídos e aplicativos móveis também podem aproveitar as vantagens do WebAssembly [Rossberg 2024].

A pesquisa sobre o formato WebAssembly no primeiro ano da proposta de desenvolvimento é bastante limitada, pois o foco estava na definição do formato. A primeira visão geral do formato é apresentada por [Haas et al. 2017]. Esta é a primeira discussão aprofundada sobre funcionalidades e opções de *design*. O artigo faz uma excelente abordagem para apresentar esta nova tecnologia, discutindo os conceitos por trás do processo de validação, execução e respectivas restrições. Sendo um dos primeiros textos sobre WebAssembly, o trabalho ajuda a comunidade com uma rica documentação e orienta numerosos trabalhos.

Algumas das principais *features* encontradas no WebAssembly em 2024 são:

- *Mutable globals*: agora variáveis *const* podem ser importadas ou exportadas, oferecendo um ambiente mais flexível para o desenvolvimento de aplicações, como, por exemplo, aplicações *multithreaded*.
- *Multiple memories*: módulos podem declarar várias memórias; no entanto, módulos e funções só podem fazer referência a uma memória por vez, impossibilitando a transferência de dados entre as memórias.
- Tratamento de exceções, tanto aquelas lançadas em módulos Wasm quanto por funções em outras linguagens que são importadas e invocadas em código WebAssembly.
- Coletor de lixo para gerenciamento de memória, que já foi adotada no Mozilla Firefox e no Google Chrome.

6.5. Aspectos de desempenho

O formato WebAssembly já é suportado pelos principais navegadores, trazendo um novo recurso para o ambiente Web ao permitir que aplicações desenvolvidas em C, C++, C#, Go e Rust sejam portadas para este ambiente [Jangda et al. 2019]. Um dos atrativos do WebAssembly é o potencial de gerar ganhos de desempenho em comparação com o JS. Diante disso, esta seção discute alguns aspectos de desempenho do WebAssembly.

Em um navegador, tanto aplicações WebAssembly quanto aplicações JS são executadas no *engine* JS. As principais diferenças seriam o modelo de execução e o controle

da memória. Uma aplicação em JS precisa passar por uma análise sintática, otimizada e compilada antes de poder ser executada. Um compilador Just-In-Time (JIT) é responsável por estas operações de otimização e geração de *bytecode* que permitem o aperfeiçoamento de códigos que serão executados várias vezes. Ao considerar a memória no JS, todo o controle é realizado pelo coletor de lixo (Garbage Collection (GC)), que realiza a identificação de blocos que não serão mais utilizados e podem ser desalocados.

Já para o formato WebAssembly o *bytecode* já está pronto, sem necessidade de realizar análise sintática ou otimização do código, já que essas etapas foram feitas durante a geração do *bytecode*. O WebAssembly armazena as informações em uma memória linear onde as informações podem ser compartilhadas entre a aplicação WebAssembly e aplicações no ambiente externo, como, por exemplo, JS. Através da memória linear, os dados são passados para um módulo WebAssembly, e as informações são compartilhadas entre JS e WebAssembly, agindo como um vetor onde os dados são armazenados. A memória é alocada através de páginas, e após alocada não pode ser desalocada. Todo o controle das informações contidas na memória linear é responsabilidade do desenvolvedor, que deve rastrear o que está sendo armazenado na memória e onde está [Battagline 2021]; quando uma linguagem de alto nível é compilada para Wasm, o compilador insere código para efetuar esse controle. Recentemente foi lançado o suporte de GC para o WebAssembly, no entanto, discussões do impacto ainda são limitadas. Versões anteriores sem o suporte do GC requerem que o desenvolvedor faça todo o processo de controle da memória linear, devido à inexistência de uma estratégia automática para recuperar memória [Yan et al. 2021].

Ao considerar o uso de memória, aplicações desenvolvidas em JS tiram o proveito do GC, que tem um direto impacto na memória utilizada pela aplicação; isso difere de aplicações WebAssembly, que possuem um uso de memória superior [Wang 2021]. Estas diferenças tornam-se consideráveis quando as entradas de dados começam a ser maiores: para o JS o uso de memória permanece constante, enquanto que as aplicações Wasm exibem crescimento do uso de memória. [Wang 2021] demonstra este comportamento com cinco tamanhos de entradas variando de pequeno a extra-grande, onde o JS mantém uma média de uso de memória de 884,42 KB contra um uso médio de 31,2 MB do WebAssembly.

Ao avaliar o tempo de execução entre WebAssembly e JS, é possível observar que o JS é impactado pelo uso de JIT, ao passo que o WebAssembly mantém um desempenho mais constante [Wang 2021, Yan et al. 2021]. Em um estudo [De Macedo et al. 2022], o WebAssembly foi 9,8% mais lento que o JS em avaliações com *benchmarks*, e foi 17,2% mais rápido na execução de aplicações reais.

O consumo de energia é um dos fatores de relevância em novas tecnologias, e o comportamento observado acaba sendo similar em relação às comparações de desempenho. Aplicações reais em WebAssembly consumiram 24,3% menos energia que aplicações em JS; no entanto, ao considerar testes com *benchmarks*, o WebAssembly gastou 5,1% mais energia que o JS [De Macedo et al. 2022].

Apesar do WebAssembly ser uma tecnologia recente, ele já está disponível em todos os principais navegadores, sendo importante as diferenças entre eles. O WebAssembly apresenta os melhores resultados ao considerar consumo energético e tempo de

execução com o Google Chrome, ao passo que com o Mozilla Firefox e Microsoft Edge os resultados variam de acordo com o tipo de aplicação [De Macedo et al. 2022]. Em uma comparação entre ambientes *desktop* e móveis [Yan et al. 2021], o Mozilla Firefox foi o navegador mais rápido para executar aplicações em JS em dispositivos móveis, e o Microsoft Edge foi o mais eficiente para WebAssembly.

Um dos *overheads* encontrados ao utilizar o WebAssembly ocorre em momentos que a aplicação requer uma troca de contexto. Ou seja, quando a aplicação está trocando o contexto entre o WebAssembly e o JS, para acessar um tipo de recurso como o DOM ou *WebSockets*. Isto ocorre em momentos que o WebAssembly necessita acessar APIs Web, já que estas operações requerem no mínimo instanciar o módulo WebAssembly [Yan et al. 2021].

[De Macedo et al. 2021] destaca que o WebAssembly foi mais rápido e teve menor consumo de energia na maioria dos testes realizados em comparação com o JS. No entanto, o trabalho também destaca que, quanto maior a entrada de dados da aplicação, menores são as diferenças de desempenho entre o WebAssembly e o JS. Em suma, apesar do WebAssembly ainda requerer melhorias para superar algumas limitações, em muitos cenários essa tecnologia já propicia ganhos de desempenho e de consumo de energia em comparação com o JS.

6.6. Aspectos de segurança em WebAssembly

No geral, a pesquisa do WebAssembly está sendo desenvolvida com diversos propósitos. As discussões atuais concentram-se na exploração da segurança desta tecnologia, na identificação de falhas de *design*, no uso deste formato com intenções maliciosas e na aplicação dos benefícios desta tecnologia a outros recursos. Além disso, atacantes também se utilizam do WebAssembly para realização de ataques, principalmente *cryptojacking* e ofuscação de códigos [Helpa et al. 2023, Heinrich et al. 2023, Heinrich et al. 2024].

Portanto, nesta seção vamos discutir como o *sandbox* do WebAssembly funciona (Seção 6.6.1), como a integridade do fluxo de controle traz mais segurança para o formato (Seção 6.6.2), como atacantes tentam explorar essa nova tecnologia e quais vulnerabilidades/riscos ainda existem e estão sendo estudados pela comunidade para a criação de contramedidas (Seção 6.6.3).

6.6.1. Sandbox WebAssembly

Uma *sandbox* é um ambiente de execução restrito e controlado que garante que *software* potencialmente malicioso, como código móvel, acesse somente os recursos do sistema para os quais esteja explicitamente autorizado [Shirey 2007]. Seja no navegador ou em um suporte de execução nativo, código WebAssembly é executado isolado dentro de uma *sandbox*, que concede acesso externo apenas através de uma API. Este *design* permite a rápida execução de aplicações no ambiente web, uma vez que a aplicação está pronta para ser executada após o *download* do *bytecode*. Para isso, o compilador (ou interpretador) insere verificações de tempo de execução que restringem o código à sua própria região de memória. Uma aplicação só poderá acessar informações fora da *sandbox* se a permissão correta for concedida. O módulo WebAssembly segue o Same Origin Policy (SOP) e o acesso a recursos específicos também requer o uso de uma API

[Kim et al. 2022, Johnson et al. 2023].

O Software-fault Isolation (SFI) *sandbox* permite que cada instância seja executada em seu próprio ambiente *sandbox* isolado. Desta forma, desde que o ambiente não tenha vulnerabilidades, aplicações podem ser executadas sem ameaçar a segurança do ambiente [Bosamiya et al. 2022]. Ao considerar a memória linear, ela não compartilha o mesmo espaço do código, estrutura e pilha, não podendo ser usada para corromper o ambiente de execução [Haas et al. 2017].

6.6.2. Integridade do Fluxo de Controle

Diferentemente de código nativo ou *bytecode* Java, WebAssembly possui apenas fluxo de controle estruturado (Structured Control Flow (SCF)), com as instruções em uma função organizadas em blocos bem aninhados. As aplicações devem declarar todas as funções acessíveis e seus tipos associados no momento da inicialização, mesmo quando são utilizadas bibliotecas de vinculação dinâmica. Assim, o WebAssembly garante a integridade do fluxo de controle por meio da semântica de execução do próprio formato, definindo construções de código válidas e como o fluxo de controle só pode pular para o início de uma construção válida [Dejaeghere et al. 2023, Lehmann et al. 2020].

A semântica de execução garante implicitamente a segurança de chamadas diretas por meio do uso de índices de seção de função explícitos. Instruções do tipo `goto` ou saltos para endereços arbitrários não são possíveis. Já a segurança do retorno da função é garantido por meio de uma pilha de chamadas protegida. A assinatura de tipo de chamadas de função indiretas já é verificada em tempo de execução, implementando efetivamente a integridade do fluxo de controle baseada em tipo de granularidade grossa para chamadas de função indiretas [Lehmann et al. 2020].

Ataques diretos de injeção de código não são possíveis em WebAssembly, já que não se pode executar dados na memória como instruções de *bytecode*. Entretanto, um atacante pode sequestrar o fluxo de controle de um módulo WebAssembly usando ataques de reutilização de código contra chamadas indiretas. Ressalta-se que ataques do tipo Return-Oriented Programming (ROP) convencionais usando sequências curtas de instruções não são possíveis em WebAssembly, porque a integridade do fluxo de controle garante que os alvos de chamada sejam funções válidas declaradas no tempo de carga. Entretanto, condições de corrida, como vulnerabilidades do tipo *time of check to time of use* (TOCTTOU), são possíveis no WebAssembly, uma vez que nenhuma garantia de execução ou agendamento é fornecida. Também podem ocorrer ataques *side-channel*, como ataques de temporização contra módulos.

6.6.3. Vetores de ataques e mitigações

Do ponto de vista de um invasor, o WebAssembly pode ser explorado para realizar uma ampla gama de estratégias de ataque. Nesta seção, focamos em estudos que consideram o uso do WebAssembly como vetor de ataque.

O primeiro estudo em grande escala para avaliar a presença e popularidade do WebAssembly na Internet foi realizado por [Musch et al. 2019]. Ele apontou que 50% das páginas identificadas como usando WebAssembly acabaram explorando esse recurso

para alguma atividade maliciosa (no top 1 milhão do Alexa[3], 56% das páginas estavam usando WebAssembly para fins maliciosos, como ofuscação ou *cryptojacking*) e apenas 10% das páginas observadas tinham códigos únicos. Portanto, antes de mergulhar nas falhas e vulnerabilidades do WebAssembly, discutiremos os tipos de vetores de ataque explorados pelos invasores.

Com foco nos binários WebAssembly (*.wasm*) que podem ser encontrados na Internet, o artigo [Hilbig et al. 2021] coletou e avaliou as propriedades de segurança de 8,4 k binários. Eles identificaram que binários gerados por linguagens que não possuem garantias de segurança na memória podem exportar essas vulnerabilidades para WebAssembly; a importação de API potencialmente perigosa existia em 21% dos binários observados, e 65% dos binários observados estavam usando uma parte da memória linear que não era gerenciada.

6.6.3.1. *Cryptojacking*

Cryptojacking é um ataque que visa explorar os recursos computacionais das vítimas para minerar criptomoedas. O WebAssembly é atrativo para esse tipo de operação devido ao ganho de desempenho em comparação a outras linguagens Web, uma vez que o desempenho impacta diretamente nos ganhos obtidos com esse tipo de ataque.

WebEth [Tiwari et al. 2018] é uma arquitetura distribuída de *cryptojacking* que usa JS e WebAssembly para mineração de Ethereum em um conjunto de navegadores. Sem a exigência de dependências externas, a implementação lida com a restrição de memória e rede imposta aos navegadores. Apesar do desempenho ser 30% mais lento que uma implementação em C++, a técnica *lazy* adotada é a principal contribuição do trabalho.

[Bian et al. 2019, Bian et al. 2020] propõem o *MineThrottle*, uma ferramenta que usa funcionalidades do WebAssembly e de algoritmos conhecidos de *cryptojacking* para identificar e interromper a execução deste tipo de ataque. Uma contribuição do trabalho é a discussão de possíveis estratégias para identificação deste tipo de ataque, apresentando as estratégias existentes e seus pontos fortes e fracos. As estratégias para detecção de *cryptojacking* na realidade utilizam um perfil para penalizar o processo de mineração que poderia estar em execução. A avaliação usa Alexa top 100 mil e 1 milhão. A estratégia de impressão digital atingiu um falso negativo/positivo inferior a 2%.

Seismic é um sistema de detecção que avisa os usuários quando uma atividade de *cryptojacking* é detectada [Wang et al. 2018]. A detecção baseada em assinatura é facilmente evitada por um invasor com o uso de estratégias de ofuscação. O *Seismic* utiliza recursos de código semântico para o processo de detecção, atingindo uma precisão de 98% para quatro famílias de algoritmos de *cryptojacking* em WebAssembly.

MinerGate é uma defesa baseada em aprendizado de máquina contra *cryptojacking* [Yu et al. 2020], que usa uma extensão de *proxy* para proteger o navegador. A extensão foi treinada com um conjunto de algoritmos de mineração em WebAssembly e é capaz de identificar elementos de interação de uma atividade de *cryptojacking* na rede.

[3]O ranking Alexa consiste em um conjunto de dados onde os sites são classificados de acordo com seu tráfego [Le Pochat et al. 2019].

[Petrov et al. 2020] apresenta *CoinPolice*, um método de detecção baseado em um classificador de rede neural profunda. Os recursos utilizados para o modelo consistem em características de desempenho e padrões de execução. A proposta detectou 97,8% dos mineradores e teve um falso positivo de 0,7%. No conjunto de dados reais, foram identificadas 6,7 mil páginas com mineradores ocultos.

6.6.3.2. Ofuscação

Ofuscação visa a transformação do código fonte ou binário com o objetivo de alterar sua aparência. Isto pode ser usado para a proteção da propriedade intelectual do código ou para ocultar código com fins maliciosos [Balakrishnan and Schulze 2005]. Como o WebAssembly é uma nova tecnologia na Web, usuários maliciosos podem tentar usá-lo para contornar medidas de segurança por meio de ofuscação de código.

[Romano et al. 2022] apresenta uma nova estratégia para escapar dos processos clássicos de identificação já propostos para JS através do uso de ofuscação de código. *Wobfuscator* é uma ferramenta de ofuscação que visa utilizar WebAssembly para movimentar parte do processamento que seria realizado em JS. Desta forma, a aplicação ainda atingirá seu objetivo, mas com pontos-chave da implementação sendo migrados para WebAssembly é possível fugir das soluções de análise estática já propostas para identificação de ataques em JS.

Um estudo do potencial de ofuscação através do uso do WebAssembly é o foco de [Bhansali et al. 2022]. Através de uma série de amostras benignas e maliciosas, diferentes estratégias de ofuscação foram aplicadas e testadas contra um detector de *cryptojacking*. Os resultados mostraram uma oportunidade altamente eficaz no uso de ofuscação para evitar engenharia reversa e descompilação do binário Wasm. O sucesso da ofuscação é influenciado pelo tipo de aplicação e pela complexidade do código, onde o código com características específicas dificultam o processo de ofuscação.

[Wang et al. 2019] apresenta uma solução para ofuscar operações numéricas intensivas encontradas em JS. Este sistema, denominado *JSPro*, consiste em uma virtualização de código para JS construído em WebAssembly. No entanto, o processo de tradução é limitado e, em alguns casos, os aplicativos não puderam ser traduzidos corretamente para o WebAssembly.

[Arteaga et al. 2021] aborda a diversificação de código para binários WebAssembly. Esta estratégia visa a geração de diferentes binários para um mesmo programa. O estudo apresenta *CROW*, um *framework* para diversificação de código em WebAssembly. A solução permite a diversificação de binários e rastreamentos para um programa. O *framework* é responsável por automatizar fluxos de trabalho para LLVM[4]. Os resultados mostram que a proposta foi capaz de alcançar diversidade ao gerar código para 79% do total de amostras no conjunto de dados utilizado.

[4]O projeto LLVM oferece um conjunto de compiladores e conjuntos de ferramentas que podem ser usados para o desenvolvimento de qualquer linguagem de programação [LLVM-Team 2023].

6.6.3.3. *Write Primitive*

Write Primitive explora vulnerabilidades encontradas em linguagens, como C, que podem permitir que um invasor obtenha uma primitiva de gravação no WebAssembly. Em alguns casos, esta vulnerabilidade não está presente na linguagem de origem, mas aparece quando o código é portado (compilado) para Wasm [Lehmann et al. 2020]. Na literatura, algumas vulnerabilidades utilizadas para alcançar esse propósito são:

- *Integer overflows/underflows:* para armazenar variáveis, o formato WebAssembly suporta tipos numéricos estáticos, semelhantes a linguagens como C/C++ e Rust. Um tipo esperado precisa ser definido com a variável que armazenará o valor. O uso de tipagem estática oferece melhor desempenho, pois não há necessidade de rastrear os tipos de variáveis durante a execução. Linguagens como Python e JS possuem tipagem dinâmica, onde o tipo de uma variável é assumido durante a execução (desta forma o desenvolvedor não precisa se preocupar com o tipo da variável, e este tipo pode mudar durante a execução). No entanto, quando WebAssembly e JS interagem entre si, um valor inesperado pode ser enviado do JS para WebAssembly. Um exemplo seria a criação de um valor para uma variável que está fora do intervalo de representação no WebAssembly, possivelmente resultando em um *overflow* de inteiro. Isso poderia ser explorado para realizar um *buffer overflow* [McFadden et al. 2018].
- *Stack Overflow:* um exemplo deste ataque no WebAssembly pode ser alcançado no caso de recursão excessiva ou violação nas suposições internas da pilha. Em vez de uma falha, os invasores poderão sobrescrever os dados, uma vez que a proteção na pilha não gerenciada não existe no WebAssembly [Lehmann et al. 2020].
- *Heap Metadata Corruption:* o invasor explora o alocador de memória que foi usado no binário Wasm, uma vez que esta informação é armazenada com o binário [Lehmann et al. 2020].

Compilar um programa de uma determinada linguagem para um binário Wasm é conhecido como portar um programa. Considerando uma perspectiva de segurança, alguns questionamentos aparecem na propagação de vulnerabilidades da linguagem fonte e no surgimento de um comportamento inesperado. Isso ocorre devido a diferentes opções de *design* entre eles. [Stiévenart et al. 2022b] apresenta descobertas sobre programas que travariam na linguagem de origem, mas seriam executados em WebAssembly e diferentes tipos para variáveis.

[Romano et al. 2021] apresenta um estudo sobre *bugs* em compiladores WebAssembly. Como as aplicações WebAssembly tendem a depender de um compilador que converte código de outra linguagem, lidar com os *bugs* torna-se um problema ainda mais complexo. Um total de nove desafios exclusivos para compiladores WebAssembly foram identificados ao considerar o desenvolvimento para WebAssembly. Eles são *Asyncify Synchronous Code*, *Incompatible Data Types*, *Memory Model Differences*, *Bugs in Other Infrastructures*, *Emulating Native Environment*, *Supporting Web APIs*, *Cross-Language Optimizations*, *Runtime Implementation Discrepancy*, e *Unsupported Primitives*. Esses

desafios foram encontrados após investigar o compilador Emscripten, e a maioria dos *bugs* está relacionada à manipulação de *strings* e arquivos.

No geral, ao considerar o compilador e a portabilidade da aplicação, são possíveis três vetores de ataque no WebAssembly. A primeira seria explorar vulnerabilidades em linguagens que serão compiladas no Wasm. Essa estratégia permite que invasores desencadeiem um comportamento inesperado[5] e permite acesso a recursos inacessíveis no WebAssembly. O segundo vetor são as vulnerabilidades encontradas no compilador, essas vulnerabilidades são devidas a recursos particulares encontrados em cada implementação. O terceiro vetor são as vulnerabilidades encontradas no WebAssembly, que existem devido ao seu *design*. Esses vetores poderiam ser combinados para efetuar um ataque e, conforme discutido, uma série de vulnerabilidades e falhas já estão mapeadas.

6.6.3.4. *Overwriting Data*

Overwriting Data é uma primitiva de ataque que permite que invasores sobrescrevam dados de forma que controle adicional seja concedido ao ator mal-intencionado durante a execução do aplicativo [Lehmann et al. 2020]. Algumas das vulnerabilidades usadas para atingir essa capacidade de sobrescrever dados no WebAssembly são:

- *Format String:* se um invasor controlar o formato de uma função `print`, ele poderá ler ou escrever na memória linear. Isso acontece pela falta de suporte a alguns formatos, o que impacta diretamente na operação da função [McFadden et al. 2018].
- *Stack-Based Buffer Overflows:* um *runtime* WebAssembly possui proteções para bloquear aplicativos que escrevem fora dos limites alocados para a memória linear [McFadden et al. 2018]. No entanto, não possui proteção contra a substituição de variáveis encontradas dentro desses limites. As variáveis podem ser substituídas pelo uso de uma função insegura [Lehmann et al. 2020].
- *Cross Site Scripting (XSS):* como as variáveis armazenadas na memória linear podem ser substituídas, as variáveis estáticas não são armazenadas com segurança. As informações poderiam ser escritas sobre posições já ocupadas na memória linear, permitindo um ataque XSS. A memória não é o único vetor para XSS; os invasores também podem controlar ponteiros de função [McFadden et al. 2018].
- *Overwriting Stack Data:* como nenhum mecanismo de segurança está presente em uma pilha não gerenciada, o invasor pode explorar a memória linear sobrescrevendo os dados da função. Porém, o alcance deste ataque é limitado, uma vez que os endereços de retorno não estão presentes na pilha e apenas as chamadas ativas serão exploradas [Lehmann et al. 2020].
- *Overwriting Heap Data:* o *design* de memória no WebAssembly coloca a *heap* após a pilha, sem nenhum mecanismo de defesa para evitar ataques. A corrupção do *heap* é apenas uma questão de estourar a pilha [Lehmann et al. 2020].

[5]Consistem em comportamentos que podem ser desencadeados devido à portabilidade do aplicativo ou interações com outros recursos no ambiente [Lehmann et al. 2020].

- *Overwriting Constant Data:* o *design* da memória linear torna impossível definir uma variável imutável. Um dado constante pode ser substituído ou alterado por um *stack-overflow* ou *stack-based buffer overflow* [Lehmann et al. 2020].

- *Redirect Indirect Calls:* através da substituição da memória linear, um invasor pode alterar um índice que poderia ser usado para emitir uma chamada indireta de função [Lehmann et al. 2020].

- *Code Injection:* permite que invasores façam alterações indesejadas no ambiente. Um exemplo seria no *host*, onde o código pode ser adicionado para substituir argumentos encontrados no aplicativo WebAssembly.

6.6.3.5. *Side-channel Attacks*

Side-channel attacks exploram erros de implementação que permitem que invasores acessem e extraiam informações de um aplicativo. Diferentes estratégias são exploradas de acordo com a aplicação e o ambiente [Spreitzer et al. 2017, Genkin et al. 2018]. A invocação de funções presentes no Emscripten poderia permitir a execução de código no lado do servidor [McFadden et al. 2018].

Uma demonstração de ataques *side-channel* no WebAssembly é apresentada por [Genkin et al. 2018]. O ataque não depende de vulnerabilidade no navegador, explorando o acesso à memória de fora da *sandbox*. O ataque foi feito contra bibliotecas criptográficas através de um código portátil.

Ataques de *timing side-channel* é o foco de [Mazaheri et al. 2022]. Apesar do estudo focar em JS, são apresentadas contramedidas para esses tipos de ataques em JS e WebAssembly. O ataque de *cache* e o ataque *Spectre* são as duas estratégias usadas para realizar o ataque de *side-channel*. As contramedidas são apresentadas com uma abordagem de detecção chamada *Lurking Eyes*, que aplica correções nos níveis de *hardware* e *software*.

6.6.3.6. Classificação

A Tabela 6.1 apresenta uma visão geral dos vetores de ataque discutidos anteriormente, com a respectiva categoria de ataque, a referência onde esta ameaça foi apresentada e referências que propõem contramedidas. As soluções de contramedidas são poucas e têm limitações na hora de serem aplicadas. Para os invasores, o WebAssembly oferece, além de diversas vulnerabilidades, flexibilidade na utilização do *design* do formato WebAssembly para realizar ataques.

6.7. Estratégias de mitigação, monitoramento e detecção de ameaças

Na literatura encontramos tanto propostas que visam melhorar a segurança do ambiente WebAssembly (Seção 6.7.1), como recursos para realizar a análise e detecção de ameaças no ambiente (Seção 6.7.2). Esta seção visa contextualizar a respeito destes tópicos.

Tabela 6.1. Visão geral das falhas e vulnerabilidades do WebAssembly usadas em ataques

Categoria	Ameaças	Contramedidas
Buffer Overflows		[Michael et al. 2023]
Cross Site Scripting (XSS)	[McFadden et al. 2018, Lehmann et al. 2020]	
Format String	[McFadden et al. 2018]	
Heap Metadata Corruption	[Lehmann et al. 2020]	
Integer Overflow/Underflow	[McFadden et al. 2018]	[Michael et al. 2023]
Stack Based Buffer Overflows	[McFadden et al. 2018, Lehmann et al. 2020]	
Stack Overflow	[Lehmann et al. 2020]	
Overwriting Constant Data	[Lehmann et al. 2020]	
Overwriting Heap Data	[Lehmann et al. 2020]	
Overwriting Stack Data	[Lehmann et al. 2020]	
Server-side Remote Code Execution	[McFadden et al. 2018]	
Side-Channel	[Genkin et al. 2018]	[Narayan et al. 2021, Protzenko et al. 2019]
Redirect Indirect Calls	[Lehmann et al. 2020]	
Code Injection	[Lehmann et al. 2020]	
Constant-Time		[Mazaheri et al. 2022, Tsoupidi et al. 2021, Watt et al. 2019a, Renner et al. 2018]

6.7.1. Melhorando a segurança do WebAssembly

As melhorias na segurança do WebAssembly abrangem estudos que introduzem correções para falhas anteriores de *design* e/ou que adicionam recursos de segurança ao formato e seu ambiente.

[Bosamiya et al. 2022] explora duas técnicas de *sandbox* para WebAssembly, avaliando a segurança e o desempenho das propostas. A primeira técnica usa provas verificadas por máquina para garantir que todas as entradas sejam seguras para serem executadas em uma *sandbox*. A segunda técnica utiliza garantias de segurança comprováveis, aproveitando as salvaguardas presentes no *design* do Rust para trazer segurança e desempenho ao compilador. O tempo de execução não tem impacto no desempenho e a segurança foi aprimorada.

RLBox é um *framework* para *sandbox* de bibliotecas de terceiros que podem oferecer algum perigo à segurança do navegador [Narayan et al. 2020]. Foram avaliados dois mecanismos de isolamento: SFI e baseado em processos multi-core. O impacto da solução no desempenho é modesto, sendo o valor mais alto de 13% na latência e 25% no uso de memória.

Spectre é um ataque que pode contornar as medidas de segurança do WebAssembly [Kocher et al. 2020]. *Swivel* é um *framework* desenvolvido para proteger contra esse tipo de ataque, fortalecendo a segurança da *sandbox* WebAssembly contra ataques de *breakout* e *poisoning* [Narayan et al. 2021]. As propostas têm uma sobrecarga, porém oferecem um ambiente *sandbox* mais seguro usando um isolamento de memória mais

forte.

Alguns autores exploraram o uso de ambientes confiáveis de execução, como Intel SGX [Will et al. 2021], para melhorar a segurança do ambiente de execução WebAssembly:

- *AccTEE* é um *framework sandbox* bidirecional que oferece segurança em casos de computação remota [Goltzsche et al. 2019]. A proteção existe com o uso de Software Guard Extensions (SGX) e *sandbox* WebAssembly para proteger a execução e os dados no ambiente. Além de garantir integridade e confidencialidade, o modelo de solução oferece proteção contra códigos maliciosos e não confiáveis.
- TWINE é um *design* de suporte de tempo de execução que explora SGX e a interface WASI para criar um ambiente confiável para executar aplicativos não modificados que já foram compilados para WebAssembly [Ménétrey et al. 2021]. A proposta é um Trusted Execution Environment (TEE) que utiliza o *sandbox* WebAssembly para abstrair o ambiente da aplicação. Esta proposta traz um ambiente de execução mais seguro para *cloud computing*, além de apresentar uma comparação detalhada de desempenho.
- *Se-Lambda* é uma estrutura de computação *serverless* que se concentra na segurança no ambiente de *cloud computing* [Qiang et al. 2018]. A solução *sandbox* bidirecional protege o *gateway* API, os dados do usuário e a plataforma na Cloud. O *framework* apresenta baixo impacto no desempenho e, como o protótipo é baseado em um projeto de código aberto, novas funcionalidades poderiam ser adicionadas acompanhando novas melhorias no WebAssembly.
- [Pop et al. 2022] apresenta um *design* de enclave baseado em WebAssembly. A proposta garante integridade e confidencialidade, oferecendo um canal seguro para os serviços Wasm migrarem entre diferentes arquiteturas.
- *WaTZ* é um *runtime* WebAssembly baseado em TEE que também oferece suporte a atestação remota [Ménétrey et al. 2022]. O isolamento da *sandbox* WebAssembly evita ataques de elevação de privilégios, com algum impacto no desempenho.
- *Edgedancer* é uma plataforma para *edge computing* móvel que utiliza WebAssembly para garantir segurança e permite a automigração dentro da infraestrutura [Nieke et al. 2021]. A plataforma também aproveita o TEE para melhorar a segurança. Pequenos serviços apresentam melhor desempenho durante a migração com maior padrão de segurança.

Fuzzing é uma técnica de teste de *software* que visa encontrar pontos fracos, falhas e vulnerabilidades de *design* por meio da geração de testes de entrada [Liang et al. 2018]. Esta estratégia permite monitorar e estudar o comportamento de uma aplicação:

- *WAFL* é uma ferramenta de *fuzzing* para binários WebAssembly que usa AFL++, um *fuzzer* bem conhecido [Haßler and Maier 2021]. A solução conecta um aplicativo WebAssembly que está sendo testado com o AFL++. A solução não prejudica o desempenho dos compiladores.

- *Fuzzm* é um *fuzzer* para formato WebAssembly, focado principalmente na detecção de vulnerabilidades na memória [Lehmann et al. 2021]. Através da implementação de canários, o *fuzzer* usa entradas do AFL para detectar *overflows* e *underflows* de memória que podem ser explorados na pilha e no *heap*. O *fuzzer* requer apenas o binário.

- [Chen et al. 2022] apresenta o *WASAI*, um *fuzzer* para identificação de vulnerabilidades em contratos inteligentes implementados em WebAssembly. O estudo constatou que, em um conjunto de 991 amostras, mais de 70% apresentavam algum tipo de vulnerabilidade.

A análise *taint* rastreia o fluxo de entrada de dados não confiáveis através de um aplicativo durante a execução [Ming et al. 2016]. Esta estratégia é importante para verificar violações de políticas e operações inseguras.

- O rastreamento *taint* de aplicações Wasm é apresentado em [Szanto et al. 2018]. Através de uma máquina virtual WebAssembly implementada em JS, o *framework* permite uma estratégia de rastreamento de *taint* para o estudo da sensibilidade dos dados do *bytecode*. A avaliação descreve uma sobrecarga de memória aceitável, com a estrutura sendo limitada apenas por algumas funcionalidades ausentes apresentadas no WebAssembly.

- *TaintAssembly* é um *framework* para rastreamento de *taint* em WebAssembly, permitindo o estudo das interações entre WebAssembly e JS [Fu et al. 2018]. A estrutura tem sobrecarga de desempenho, sofre de limitações de *design* que exigem geração de números aleatórios e não implementa operações de comparação.

SELWasm é uma estrutura para proteger a propriedade intelectual do código WebAssembly [Sun et al. 2019]. A estrutura protege contra a reutilização de código-fonte sem autorização, usando criptografia para garantir que os invasores não consigam obter o código-fonte e uma estratégia de *lazy loading* para otimização.

BLADE é um *framework* para proteção contra vazamentos de especulação de algoritmos criptográficos [Vassena et al. 2021]. O *framework* é implementado no compilador WebAssembly e permite a avaliação de sua implementação através de implementações vulneráveis do WebAssembly. A solução não requer supervisão e pode ser eficaz em outros linguagens.

Com o objetivo de entender as vulnerabilidades por trás da tecnologia WebAssembly para aplicações multiplataforma, o trabalho [André et al. 2022] foca nas dúvidas dos desenvolvedores encontradas no StackOverflow[6] para entender melhor os problemas de segurança. Mais da metade das perguntas são direcionadas a informações sobre recursos ou *bugs*. A autenticação é o grupo de questões mais frequentes.

Considerando o impacto na segurança da compilação de aplicações de C para WebAssembly, e explorando que tipo de comportamento poderia ser esperado, os autores

[6]StackOverflow é um site focado em perguntas e respostas [stackoverflow 2023].

em [Stiévenart et al. 2021] apresentam um estudo com foco na avaliação das vulnerabilidades/divergências que são portadas de C após o processo de compilação ser concluído. Em um conjunto de dados com 4,4 mil amostras, 24% dos aplicativos testados tiveram um resultado diferente, devido à falta de medidas de segurança como canários de pilha quando portados para WebAssembly.

6.7.2. Análise e proteções para WebAssembly

A análise de aplicações WebAssembly permite que os desenvolvedores tenham um melhor entendimento do ambiente em que as aplicações serão executadas, oferecendo informações sobre interações com outras linguagens, impactos no desempenho e segurança. Medidas de proteção poderiam ser tomadas após este tipo de estudo.

[Romano and Wang 2020a] apresenta *WASim*, uma ferramenta para classificação de módulos WebAssembly. Onze categorias foram usadas para treinar quatro modelos de aprendizado de máquina. O conjunto de dados foi extraído do Alexa 1 milhão, com os modelos atingindo uma precisão de 91,6%.

[Helpa et al. 2023, Helpa et al. 2024] apresenta uma estratégia para a detecção de anomalias em binários WebAssembly. A proposta extrai características dos binários através do uso do formato Debugging With Attributed Record Format (DWARF), sendo um formato para depuração que permite acesso a seções específicas do binário.

[Heinrich et al. 2023, Heinrich et al. 2024] apresentam duas estratégias de detecção de anomalias para aplicações WebAssembly. As estratégias coletam chamadas WASI realizadas pelo ambiente *sandbox* do WebAssembly, posteriormente utilizando estes dados em uma estratégia de categorização que permite a detecção de anomalias das aplicações.

WebAssembly Symbolic Processor (WASP) é o foco em [Marques et al. 2022]. Essas estratégias permitem que os desenvolvedores encontrem *bugs* e falhas de segurança por meio da análise de múltiplos caminhos de programas.

Oron é uma plataforma de instrumentação que foi implementada usando WebAssembly para resolver problemas de desempenho do *Linvail* [Munsters et al. 2021]. A plataforma foi avaliada através da instrumentação do código AssemblyScript. As avaliações mostram menos sobrecarga de desempenho em comparação com outras soluções.

Wasmati é uma ferramenta de análise estática apresentada por [Brito et al. 2022], com foco em encontrar vulnerabilidades em binários Wasm. *Wasmati* foi desenvolvido para ser utilizado na fase de desenvolvimento, limitando-se a binários baseados em Emscripten. A técnica utilizada para a avaliação consiste em Code Property Graph (CPG) que inclui informações sobre o código que está sendo analisado como pares propriedade-valor.

[Lehmann and Pradel 2019] apresenta *Wasabi*, que é uma estrutura para avaliação dinâmica de WebAssembly. A primeira contribuição consiste em um levantamento de abordagens com análise dinâmica encontradas na literatura. *Wasabi* aproveita a implementação do WebAssembly para extrair informações em tempo de execução, além de não se limitar apenas a aplicações implementadas em WebAssembly.

[Stiévenart and De Roover 2020] analisa binários Wasm por meio do isolamento

de segmentos do código. A avaliação se concentra no desenvolvimento de uma análise de fluxo das aplicações WebAssembly. Apesar de uma precisão baixa de 64%, a contribuição do artigo consiste em uma análise estática que permite a avaliação de funcionalidades do código WebAssembly.

EOSafe é uma estrutura que analisa o binário Wasm para encontrar vulnerabilidades de contratos inteligentes EOSIO[7] [He et al. 2021]. A estratégia de detecção depende de heurísticas e da limitação da execução simbólica. Quatro das vulnerabilidades mais populares para EOSIO são exploradas, com os modelos alcançando um *f1score* de 98%.

WasmView é um *framework* para avaliar/visualizar a interação entre JS e WebAssembly [Romano and Wang 2020b]. O *framework* permite o entendimento de chamadas de função entre as linguagens e *stack traces* das aplicações, através da apresentação de um gráfico visual de chamadas e *trace logs* das informações capturadas.

6.8. Tendências recentes e problemas abertos de pesquisa

Para compreender as tendências e problemas em aberto ao considerar o formato WebAssembly, são consideradas seis das principais características na literatura, estas sendo as novas propostas para aperfeiçoar o formato (Seção 6.8.1), tipos de aplicações em que o WebAssembly está sendo adotado (Seção 6.8.2), conjuntos de dados que estão disponíveis e podem ser utilizados para o estudo do WebAssembly (Seção 6.8.3), as limitações encontradas (Seção 6.8.4), os tópicos de pesquisa em aberto (Seção 6.8.5), e o estado atual de desenvolvimento (Seção 6.8.6).

6.8.1. Novas propostas para o formato WebAssembly

Conforme discutido na Seção 6.6.3, apesar do foco na segurança por trás do *design* do WebAssembly, falhas e vulnerabilidades existem e podem ser exploradas por usuários mal intencionados. Porém, existe um conjunto de trabalhos que visam propor melhorias de *design* para o formato WebAssembly, apresentando contramedidas e propostas para aumentar a segurança no ambiente. Esta seção tem como objetivo discutir esses trabalhos.

Constant-Time WebAssembly (CT-Wasm) é uma expansão do compilador WebAssembly, com foco principal em segurança criptográfica [Watt et al. 2019a]. CT-Wasm permite a verificação de propriedades de segurança e é seguro contra ataques de *side-channel*. A proteção de dados permite que os desenvolvedores melhorem a segurança considerando o fluxo de informações. [Renner et al. 2018] também foca na limitação de segurança do tempo constante no WebAssembly para primitivas criptográficas. Para atingir este objetivo, o compilador WebAssembly foi modificado, adicionando-se o suporte à sensibilidade de um tipo de variável. A ampliação do processo de validação com adição de funcionalidades de verificador de tipo, e o ambiente de execução foi modificado para garantir a segurança. A verificação de primitivas criptográficas no WebAssembly é o foco do [Protzenko et al. 2019]. Um conjunto de ferramentas para compilar F*[8] para WebAssembly é implementada, o que oferece a opção de usar bibliotecas de plataforma via WebAssembly. O processo de validação consiste em compilar HACL*[9] para WebAs-

[7]EOSIO é um protocolo blockchain [EOSIO 2023].

[8]F* é uma linguagem de programação de uso geral orientada para provas [fstar-lang 2023].

[9]HACL* é uma biblioteca criptográfica escrita em F* [HACL* 2023].

sembly e verificar a proteção contra ataques de *side-channel*.

Gobi é um SFI para WebAssembly, que visa resolver as limitações do *sandboxing* do WebAssembly [Narayan et al. 2019]. O sistema agrupa compiladores e alterações no tempo de execução que permitem que bibliotecas implementadas em linguagens como C/C++ sejam compiladas para WebAssembly. Este protótipo do WebAssembly SFI foi incorporado pela comunidade Wasm através da criação do WebAssembly System Interface (WASI) e melhorias no conjunto de ferramentas Lucet (no entanto, em 2020 o foco foi alterado para utilizar o Wasmtime [BytecodeAlliance 2023]) .

A contribuição de [Watt et al. 2019b] está diretamente ligada ao desenvolvimento do padrão Wasm, apresentando extensões de memória e operação para WebAssembly e JS. As extensões WebAssembly permitem o uso de *threads*, atômicos e mutabilidade. O modelo de memória garante o uso sequencial de dados e a simultaneidade de memória compartilhada é garantida para implementações futuras.

O WebAssembly possui problemas relacionados à memória que ainda ocorrem dentro do *sandbox* (Seção 6.6.3). Memory-safe WebAssembly (MSWasm) é uma extensão para segurança de memória apresentada por [Michael et al. 2023]. A especificação formal define o uso de memória segmentada para segurança de memória. Duas implementações são discutidas, a MSWasm para segurança de memória em compiladores WebAssembly e um compilador C para MSWasm que garante segurança de memória contra fontes inseguras. A extensão quando avaliada apresentou um *overhead* de 197,5%.

[Vassena and Patrignani 2020] discute problemas de memória encontrados no WebAssembly e aponta soluções conhecidas que podem ser exploradas para solucionar estas limitações. Através da utilização da extensão MS-Wasm, a proposta pode garantir a segurança da memória.

Uma extensão do WebAssembly é apresentada em [Disselkoen et al. 2019], que apresenta um *design* para garantir a segurança espacial e temporal e a integridade do ponteiro. Com segurança de memória explícita no nível da linguagem, a implementação proposta do WebAssembly garante segurança de memória.

[Kolosick et al. 2022] faz modificações no *sandbox* WebAssembly para garantir melhor desempenho e segurança. O *sandbox* Wasm usa transições que são pesadas durante o tempo de execução, mudar isso para condições de custo zero também atinge o mesmo nível de garantia de segurança com melhor desempenho. Através de um verificador chamado *VeriZero* é possível identificar quando uma função é semanticamente capaz de explorar esta característica.

WebAssembly sofre com consumo de memória e tempo de inicialização, um *design* para uma solução eficaz é o foco em [Titzer 2022]. A proposta utiliza uma tabela lateral compacta gerada durante o processo de validação para fornecer melhor desempenho para o Wasm.

SecWasm é um *design* de sistema Information-Flow Control (IFC) híbrido para a confidencialidade de dados em aplicações WebAssembly [Bastys et al. 2022]. A solução supera problemas de fluxo de controle estruturado e memória linear, mas seu impacto no desempenho é desconhecido, uma vez que ela não foi implementada.

[Arteaga 2022] apresenta solução para mitigar a monocultura de *software* em WebAssembly. Randomização e execução multivariável foram propostas para os *frameworks* Code Randomization of WebAssembly (CROW) e Multi-variant Execution for WebAssembly (MEWE).

6.8.2. Aplicações variadas do formato WebAssembly

O uso do WebAssembly para resolver problemas encontrados em diversos cenários é considerado nesta seção. Como o WebAssembly oferece uma gama de recursos, diferentes aplicações podem explorar essa tecnologia para obter desempenho e ganho de segurança.

Na literatura, um conjunto de trabalhos foca na aplicação do WebAssembly para melhorar o ambiente IoT.

- [Radovici et al. 2018] propõe uma estrutura que se concentra na execução de *bytecodes* em uma *sandbox* WebAssembly isolada. A proposta depende principalmente do ambiente WebAssembly, com foco em um *design* para expor com segurança os recursos necessários para executar uma aplicação. Espera-se que a proposta contribua para a segurança e desempenho no cenário IoT.

- *Aerogel* é uma estrutura para proteger melhor o controle de acesso entre aplicações *bare-metal* e IoT usando o ambiente WebAssembly, abordando lacunas de segurança anteriormente não abordadas [Liu et al. 2021]. A estrutura utiliza o *sandbox* WebAssembly para aplicar medidas de controle de acesso, com uma sobrecarga de desempenho de 1% e o crescimento do consumo de energia atingindo quase 46%.

- *Wasmachine* é um sistema operacional (SO) focado principalmente na segurança do WebAssembly para a execução de aplicações em IoT e dispositivos de *fog computing* [Wen and Weber 2020]. Através da execução do binário WebAssembly no *kernel mode*, o custo de execução é reduzido. O artigo também implementa o *kernel* do sistema operacional em Rust para garantir a segurança da memória.

- *ThingSpire* é um SO de *cloud-edge* baseado em WebAssembly [Li et al. 2021]. A proposta aborda três desafios relacionados ao desenvolvimento da infraestrutura; (i) como projetar o ambiente de forma eficaz; (ii) como ativar a intercomunicação entre módulos; e (iii) como a segurança e a tolerância a falhas são suportadas.

- *MEWE* é um *framework* que apresenta uma técnica para *edge-cloud computing* [Arteaga et al. 2022]. A estrutura adiciona randomização para execução do tempo de execução do WebAssembly e diversifica os binários do Wasm. A combinação dessas duas estratégias fortalece ataques como Break-Once-Break-Everywhere (BOBE). Com dados reais, foi possível identificar flexibilidade para gerar variantes binárias.

Como *edge computing* possui uma demanda por desempenho, [Koren 2021] apresenta uma prova de conceito do uso de WebAssembly para microcontroladores. Um servidor web de um Integrated Development Environment (IDE) foi desenvolvido para entregar novos módulos para dispositivos *edge computing* e *cloud*.

WebCloud é uma solução de criptografia para comunicação entre serviços em Cloud e navegadores [Sun et al. 2022]. A aplicação *client-side* resolve limitações relacionadas à segurança e controle de acesso de soluções anteriores.

[Baumgärtner et al. 2019] propõe uma aplicação de WebAssembly no contexto de Disruption-Tolerant Networking (DTN). A rede é estabelecida por meio de uma combinação de Bundle Protocol 7 e WebAssembly. Uma aplicação web habilitada pelo navegador realiza a troca de mensagens.

Sledge é um *framework serverless* que utiliza WebAssembly para *edge computing* [Gadepalli et al. 2020]. Oferece isolamento leve de funções em tempo de execução implementado em WebAssembly e políticas de agendamento para a infraestrutura. Os resultados mostraram baixa latência e gerenciamento eficiente de simultaneidade.

WasmAndroid é um *runtime* multiplataforma para linguagens nativas no Android que requer apenas a compilação do código para WebAssembly [Wen et al. 2021]. Experimentos com o *benchmark* SPEC CPU 2006 mostraram que o WasmAndroid teve tempo de execução 30% superior ao da execução nativa.

WasmTree é uma implementação de Resource Description Framework (RDF) que explora WebAssembly e Rust para otimizar e ganhar desempenho no ambiente Web [Bruyat et al. 2021]. O uso do WebAssembly não apresentou ganho de desempenho. Porém, a avaliação com a consulta SPARQL apresentou melhoria de desempenho.

[Stiévenart et al. 2022a] propõe uma estratégia estática de *slicing* para programas WebAssembly. O estudo visa uma abordagem estática de *intra-procedural backward slicing* para WebAssembly. A proposta é avaliada com aplicações reais.

[Ménétrey et al. 2022] apresenta um posicionamento para o desenvolvimento de aplicações que possam rodar em um conjunto de dispositivos de *hardware* sem perda de desempenho e segurança. O estudo tem como foco descrever o impacto que o uso do WebAssembly pode trazer para esse meio.

Uma avaliação do uso de Ethereum Virtual Machine (EVM) e WebAssembly para contratos inteligentes é o foco em [Zhang et al. 2024]. O uso do WebAssembly nos clientes blockchain Ethereum apresenta problemas relacionados ao suporte, limitações de desenvolvimento e instabilidade. O desempenho do WebAssembly varia significativamente.

Hector é uma estrutura de aplicação web que permite aplicações distribuídas no navegador web [Goltzsche et al. 2020]. WebAssembly é um dos componentes que garantem integridade, confidencialidade e isolamento.

EVulHunter é uma ferramenta desenvolvida para detecção de vulnerabilidades do EOSIO WebAssembly, focada em ataques de transferência falsa [Quan et al. 2019]. Duas estratégias são utilizadas para a análise estática, funções predefinidas e operações de comparação.

EOSDFA é um *framework* para análise dos contratos inteligentes na blockchain EOSIO [Li and Zhang 2022]. O *framework* expande o sistema de implantação automática conhecido como *framework* Octopus, permitindo a avaliação de *pointer access* para fluxo de controle e análise de fluxo de dados.

ZAWA [Gao et al. 2022] é uma máquina virtual que emula *bytecodes* WebAssembly. Visando um tempo de execução mais seguro, o ZAWA aproveita o Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (ZKSNARK), para avaliar se algum vazamento está acontecendo.

6.8.3. Conjuntos de dados disponíveis

A disponibilidade de dados para experimentação ajuda os pesquisadores a comparar resultados, melhorar pesquisas publicadas anteriormente e desenvolver novas abordagens. A Tabela 6.2 apresenta alguns dos principais conjuntos de dados disponíveis atualmente que podem ser utilizados para o desenvolvimento de pesquisas com WebAssembly, descrevendo o tipo de dados, número de amostras, metodologia de extração e ano de criação.

Tabela 6.2. Datasets WebAssembly

Dataset	Propósito	Amostras	Extração de Dados	Ano
WasmBench [Hilbig et al. 2021]	Uma representação de conjunto de dados do mundo real	8.4 K	Diversas fontes, como: GitHub, gerenciador de pacotes, sites e manualmente	2021
SnowWhite [Lehmann and Pradel 2022]	Recuperação de tipos WebAssembly	6.3 M	4 k amostras de pacotes de código-fonte do Ubuntu C e C++	2022
[Chen et al. 2022]	Identificação de vulnerabilidades de contratos inteligentes Wasm	3.3 k	Vulnerabilidades e contratos inteligentes	2022
QRS [Stiévenart et al. 2021]	Aplicações C com comportamento diferente quando compilados para WebAssembly	1,088	Código C vulnerável do Juliet Test Suite 1.3 (2017)	2021
SAC [Stiévenart et al. 2022b]	Apresenta binários Wasm com problemas de segurança ao portar WebAssembly de C	4.9 K	Código C vulnerável do Juliet Test Suite 1.3 (2017)	2022
Alexa [Amazon 2023]	Uma classificação de 1 milhão dos domínios de maior tráfego em todo o mundo, atualizada diariamente	Variável	Os dados de tráfego são classificados por domínio. Os dados são coletados por meio de uma extensão do navegador usada pelos usuários	2008-2022
Baseado no Alexa [Musch et al. 2019]	Define o uso do WebAssembly na web	150	O módulo WebAssembly de páginas da web foi coletado, com 1.950 amostras (150 exclusivas) de 1.639 sites	2019
Dataset de perguntas [André et al. 2022]	Perguntas relacionadas à segurança do WebAssembly feitas por desenvolvedores	359	Perguntas de segurança do Stack Overflow	2022
Dataset de bugs [Romano et al. 2021]	Coleção de bugs do Emscripten	1,054	Bugs foram extraídos do projeto Emscripten	2021

No geral, apesar da quantidade limitada de conjuntos de dados que podem ser uti-

lizados em estudos de segurança, as amostras mais recentes pretendem representar o atual ecossistema Wasm. É importante notar que estes conjuntos de dados não estão sendo atualizados com amostras mais recentes. Isto pode ser considerado arriscado, considerando que o WebAssembly ainda está passando por grandes mudanças e que dados mais recentes e mais representativos serão necessários no futuro.

O ranking Alexa [Amazon 2023] é uma fonte recorrente de dados para experimentos não apenas em WebAssembly. Na coleta de dados apresentada por [Kim et al. 2022], apenas uma abordagem não explora o ranking Alexa[10]. No entanto, o uso do ranking Alexa para pesquisas de segurança não é adequado, uma vez que o método utilizado para coleta de dados varia muito e a falta de detalhes pode afetar os resultados da pesquisa [Le Pochat et al. 2019, Ruth et al. 2022]. Algumas alternativas mais adequadas para representar o ambiente Web para pesquisas de segurança são discutidas em [Xie et al. 2022, Le Pochat et al. 2019, Durumeric 2023].

Também é importante considerar o tipo de informação que cada conjunto de dados fornece. Para a extração de informações de binários, pode-se utilizar a maioria das amostras disponíveis nos conjuntos de dados apresentados. Porém, para uma avaliação que requer a execução dos binários, restrições relacionadas à falta de dependências do período de extração de dados ou mesmo suporte dos dados, limitam o uso desse conjunto de dados. Uma alternativa, neste caso, seria usar ferramentas de *benchmark* e conjuntos de testes que funcionam em compiladores.

6.8.4. Limitações atuais do WebAssembly

Apesar do foco na segurança, algumas opções de *design* no WebAssembly limitam as medidas de segurança encontradas em outras linguagens que podem ser compiladas para o WebAssembly. Em alguns casos, esses mecanismos não estarão presentes quando o código for compilado de outra linguagem para Wasm [McFadden et al. 2018].

Address Space Layout Randomization (ASLR) é uma proteção de memória que, apesar de estar presente em outras linguagens, seria difícil de ser implementada em WebAssembly, considerando o *design* da memória linear [McFadden et al. 2018]. No futuro, esta proteção poderá ser adicionada [W3C Community Group 2022a]. No entanto, a memória linear também elimina a necessidade de alguma proteção como canários de pilha. O *design* do WebAssembly também remove os requisitos de recursos como Data Execution Prevention (DEP), uma vez que instruções de baixo nível não terão o mesmo acesso aos recursos, portanto esta proteção não é necessária.

O *Heap Hardening* é apresentado em alguns compiladores e soluções foram propostas na literatura. Além disso, os compiladores que suportam Control Flow Integrity (CFI) podem exportar a proteção para o Wasm compilado [McFadden et al. 2018].

6.8.5. Tópicos de pesquisa em aberto

As discussões sobre vulnerabilidades do WebAssembly estão evoluindo e é possível perceber mudanças feitas pela comunidade com base nessas descobertas. Contudo, na prá-

[10]O projeto do rank da Alexa foi descontinuado em 2022, uma alternativa acaba sendo o Tranco [Le Pochat et al. 2019].

tica, o uso malicioso do WebAssembly tem sido limitado até agora. Vimos estudos sobre como as interações entre o WebAssembly e os recursos apresentados no ambiente podem ser úteis para invasores e o impacto do formato binário na ofuscação. Estudos futuros devem estar mais alinhados na avaliação do uso atual do WebAssembly por invasores, na eficácia do *sandbox* WebAssembly e nas reais vantagens em áreas específicas como IoT, Trusted Computing (TC) e *blockchains*, nas quais se pode observar a ampla aplicabilidade do WebAssembly.

Os estudos que visam descrever a adoção e o uso de aplicações WebAssembly no mundo real ainda são limitados, muitos deles não representativos do estado atual da Internet. A discussão sobre a adoção do WebAssembly, mesmo considerando o ponto de vista da segurança, também apresenta os mesmos problemas. Estes problemas estão diretamente associados (i) à utilização de bases de dados desatualizadas, que em alguns casos já se revelaram não representativas; e (ii) à falta de amostras, estratégias de validação e preocupações com a reprodutibilidade dos dados utilizados em estudos futuros. A disponibilidade é considerada, porém, a utilidade dos dados para estudos futuros não é considerada.

Atestados de medidas de segurança no formato WebAssembly não são estudados. A discussão sobre segurança na literatura poderia ser ampliada para o projeto/desenvolvimento do WebAssembly, visando um melhor entendimento da segurança do formato. Uma discussão semelhante seria útil para esclarecer o impacto da migração de algumas linguagens para o WebAssembly, onde várias discussões sobre segurança estão se concentrando.

6.8.6. Estado atual do desenvolvimento do WebAssembly

Os avanços da comunidade nos últimos anos demonstram o potencial que o formato WebAssembly pode alcançar, com constante atualizações e lançamento de novas *features* como o suporte para *tail calls* em 2023. No final de 2023 também foi lançado o suporte para o GC que está disponível no Google Chrome e no Mozilla Firefox. O suporte para o Single Instruction, Multiple Data (SIMD) também apresentado, que pode trazer ainda mais desempenho para aplicações em WebAssembly. Suporte para múltiplas memórias foi lançado em 2023, permitindo que um módulo possua mais de uma memória linear [WebAssembly 2024]. Para 2024 é esperado o lançamento do *snapshot_preview2* da WASI[11].

6.9. Considerações finais

O WebAssembly é um formato *bytecode* que vem para trazer mais segurança e desempenho para o ambiente da Web. O formato permite que desenvolvedores não só implementem aplicações diretamente utilizando o formato textual WAT, como também possibilita que linguagens de alto nível sejam utilizadas pelos desenvolvedores já que posteriormente estes códigos podem ser portados para o formato WebAssembly.

Através de um ambiente *sandbox*, aplicações desenvolvidas em WebAssembly podem ser executadas em um conjunto variado de arquiteturas, além de oferecer maior

[11] Detalhes das propostas e suas respectivas fases podem ser encontrados em [WebAssembly 2024].

segurança ao executar código de terceiros no *client-side*. A versão atual já consegue demonstrar as vantagens das principais características do modelo, apesar de ainda estar em um processo inicial de desenvolvimento e teste. Claro que limitações existem, mas as vantagens já são evidentes ao considerar maior segurança e desempenho no ambiente da Web.

O suporte para o formato WebAssembly já alcança muitas das principais linguagens, e uma variedade de compiladores estão disponíveis [Stephen 2022]. A adoção do formato também é alta, com os principais navegadores tendo suporte aos principais recursos. As propostas também apresentam o mesmo ritmo, com lançamentos constantes de novas funcionalidades. Wasmtime e Emscripten são dois dos compiladores que possuem um projeto ativo, com novos recursos sendo adicionados com frequência e comunidades ativas.

Na academia, o desempenho e a segurança do WebAssembly vêm sendo estudados de perto pela comunidade, com novos estudos sendo publicados discutindo as principais mudanças e impacto. Estes trabalhos também discutem novas propostas que podem gerar melhorias para o formato WebAssembly. Por exemplo, as medidas de segurança para o WebAssembly não estão limitadas a um problema específico. A melhoria da segurança está sendo alcançada por novas propostas para o *design* do WebAssembly, *frameworks* que melhoram a segurança, ou estratégias de avaliação para binários Wasm.

A melhoria da segurança do ambiente está sendo alcançada por meio de mudanças no *sandbox*, explorando recursos como SGX e API para fortalecer as proteções de tempo de execução do WebAssembly. Para o desenvolvimento foram desenvolvidas estratégias com *fuzzing*, *tracing* e *taint* para demonstrar como os recursos interagem em uma aplicação. Também é possível apontar o desenvolvimento de estratégias de análise e proteção para aplicações WebAssembly.

Novas propostas estão abordando os principais problemas do WebAssembly com melhorias no *design* do formato. Essas melhorias estão sendo aproveitadas pela comunidade para fazer mudanças no formato para garantir um ambiente mais seguro.

O formato também possui limitações e problemáticas que ainda precisam ser abordado pela comunidade. Problemas de segurança existirão em qualquer formato, pois *bugs*/falhas podem ocorrer devido à forma como os recursos e implementações são feitos pelos desenvolvedores. No WebAssembly, os invasores podem explorar vulnerabilidades ou falhas de *design* que facilitam operações maliciosas, apesar da segurança envolvida no *design* do WebAssembly. A maioria das questões de segurança discutidas neste capítulo está relacionada a alguma limitação ou escolha de projeto.

A memória linear é um dos destaques de *design* do WebAssembly, isolando a aplicação, pois os acessos à memória são limitados a uma região específica [Kim et al. 2022]. No entanto, algumas vulnerabilidades de memória persistem devido à falta de salvaguardas de segurança em linguagens que podem ser portadas para Wasm. Não se limitando à memória, um dos recursos mais atraentes do WebAssembly pode ser um dos mais perigosos. A portabilidade de uma variedade de linguagens impacta diretamente a segurança do binário Wasm, uma vez que uma variedade de linguagens implementam diferentes proteções que não estão presentes no WebAssembly e, em muitos casos, podem ser trans-

portadas ao gerar um binário Wasm.

As falhas/vulnerabilidades de segurança discutidas na literatura exploram diversos elementos. No entanto, a maioria desses problemas não se limita ao WebAssembly, portanto, enfatizar esse problema como uma restrição ao uso do WebAssembly não é adequado. Considerando o estado atual, foram feitos avanços para um ambiente mais seguro e foram propostas soluções para corrigir falhas/vulnerabilidades de segurança.

Malfeitores também podem explorar os recursos oferecidos pelo WebAssembly para realizar uma série de ataques. Os principais atrativos do WebAssembly para usuários maliciosos são (i) fornecer um formato binário (diferente de linguagens interpretadas, como JS), que pode ajudar na ofuscação de código; (ii) o ganho de desempenho do WebAssembly em comparação com outras linguagens no mesmo ambiente o torna atrativo para *cryptojacking*; (iii) por ser um recurso suportado pela maioria dos navegadores web e pela proximidade com JS, aplicações WebAssembly maliciosas tornam-se interessantes.

Na literatura também observamos lacunas que oportunizam futuras pesquisas. Essas lacunas incluem (i) a limitação de discussões entre a interação do WebAssembly com outros recursos encontrados no navegador; (ii) a falta de conjuntos de dados reais, que sejam representativos do contexto atual de uso do WebAssembly na Web; (iii) a definição de métricas adequadas de avaliação de desempenho ao considerar que módulos WebAssembly podem ser executados em diferentes arquiteturas, e serem portados de diversas linguagens de programação.

Agradecimentos

Este trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES), UDESC e FAPESC. Os autores também agradecem o Programa de Pós-Graduação em Informática da UFPR e a UTFPR *Campus* Dois Vizinhos.

Referências

[Adobe 2021] Adobe (2021). Adobe flash player eol general information page. `https://www.adobe.com/products/flashplayer/end-of-life.html`.

[Amazon 2023] Amazon (2023). Alexa Rank. `https://www.alexa.com/company`.

[André et al. 2022] André, P. M., Stiévenart, Q., and Ghafari, M. (2022). Developers struggle with authentication in Blazor WebAssembly. In *Proceedings of the 38th International Conference on Software Maintenance and Evolution*, pages 389–393, Limassol, Cyprus. IEEE.

[Apodaca 2020] Apodaca, R. L. (2020). First steps in webassembly: Hello world. Depth-First. `https://depth-first.com/articles/2020/01/13/first-steps-in-webassembly-hello-world/`.

[Arteaga 2022] Arteaga, J. C. (2022). *Artificial Software Diversification for WebAssembly*. PhD thesis, KTH Royal Institute of Technology, Stockholm, Sweden.

[Arteaga et al. 2022] Arteaga, J. C., Laperdrix, P., Monperrus, M., and Baudry, B. (2022). Multi-variant execution at the edge. In *Proceedings of the 9th Workshop on Moving Target Defense*, pages 11–22, Los Angeles, CA, USA. ACM.

[Arteaga et al. 2021] Arteaga, J. C., Malivitsis, O., Perez, O. V., Baudry, B., and Monperrus, M. (2021). CROW: Code diversification for WebAssembly. In *Proceedings of the Workshop on Measurements, Attacks, and Defenses for the Web*, Virtual Event. Internet Society.

[Balakrishnan and Schulze 2005] Balakrishnan, A. and Schulze, C. (2005). Code obfuscation literature survey. *CS701 Construction of Compilers*, 19.

[Bandhakavi et al. 2010] Bandhakavi, S., King, S. T., Madhusudan, P., and Winslett, M. (2010). VEX: Vetting browser extensions for security vulnerabilities. In *Proceedings of the 19th USENIX Security Symposium*, Washington, DC, USA. USENIX Association.

[Barth et al. 2008] Barth, A., Jackson, C., Reis, C., and The Google Chrome Team, . (2008). The security architecture of the Chromium browser. Technical report, Stanford University. `https://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf`.

[Bastys et al. 2022] Bastys, I., Algehed, M., Sjösten, A., and Sabelfeld, A. (2022). SecWasm: Information flow control for WebAssembly. In *Proceedings of the 29th International Static Analysis Symposium*, pages 74–103, Auckland, New Zealand. Springer.

[Battagline 2021] Battagline, R. (2021). *The Art of WebAssembly: Build Secure, Portable, High-Performance Applications*. No Starch Press.

[Baumgärtner et al. 2019] Baumgärtner, L., Höchst, J., and Meuser, T. (2019). B-DTN7: Browser-based disruption-tolerant networking via bundle protocol 7. In *Proceedings of the 6th International Conference on Information and Communication Technologies for Disaster Management*, pages 1–8, Paris, France. IEEE.

[Bhansali et al. 2022] Bhansali, S., Aris, A., Acar, A., Oz, H., and Uluagac, A. S. (2022). A first look at code obfuscation for WebAssembly. In *Proceedings of the 15th Conference on Security and Privacy in Wireless and Mobile Networks*, pages 140–145, San Antonio, TX, USA. ACM.

[Bian et al. 2019] Bian, W., Meng, W., and Wang, Y. (2019). Poster: Detecting WebAssembly-based cryptocurrency mining. In *Proceedings of the 26th Conference on Computer and Communications Security*, pages 2685–2687, London, UK. ACM.

[Bian et al. 2020] Bian, W., Meng, W., and Zhang, M. (2020). Minethrottle: Defending against Wasm in-browser cryptojacking. In *Proceedings of The Web Conference*, pages 3112–3118, Taipei, Taiwan. ACM.

[Bosamiya et al. 2022] Bosamiya, J., Lim, W. S., and Parno, B. (2022). Provably-Safe multilingual software sandboxing using WebAssembly. In *Proceedings of the 31st*

USENIX Security Symposium, pages 1975–1992, Boston, MA, USA. USENIX Association.

[Brito et al. 2022] Brito, T., Lopes, P., Santos, N., and Santos, J. F. (2022). Wasmati: An efficient static vulnerability scanner for WebAssembly. *Computers & Security*, 118:102745.

[Bruyat et al. 2021] Bruyat, J., Champin, P.-A., Médini, L., and Laforest, F. (2021). WasmTree: Web Assembly for the semantic Web. In *Proceedings of the European Semantic Web Conference*, pages 582–597, Virtual Event. Springer.

[BytecodeAlliance 2021a] BytecodeAlliance (2021a). Wasmtime. `https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-overview.md`.

[BytecodeAlliance 2021b] BytecodeAlliance (2021b). Wasmtime. `https://docs.wasmtime.dev/introduction.html`.

[BytecodeAlliance 2023] BytecodeAlliance (2023). Lucet has reached End-of-life. `https://hacl-star.github.io/`.

[BytecodeAlliance 2024] BytecodeAlliance (2024). Wasmtime. `https://github.com/bytecodealliance/wasmtime`.

[Chen et al. 2022] Chen, W., Sun, Z., Wang, H., Luo, X., Cai, H., and Wu, L. (2022). WASAI: Uncovering vulnerabilities in Wasm smart contracts. In *Proceedings of the 31st International Symposium on Software Testing and Analysis*, pages 703–715, Virtual Event. ACM.

[De Macedo et al. 2021] De Macedo, J., Abreu, R., Pereira, R., and Saraiva, J. (2021). On the runtime and energy performance of WebAssembly: Is WebAssembly superior to JavaScript yet? In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, pages 255–262. IEEE.

[De Macedo et al. 2022] De Macedo, J., Abreu, R., Pereira, R., and Saraiva, J. (2022). WebAssembly versus JavaScript: Energy and runtime performance. In *Proceedings of the International Conference on ICT for Sustainability*, pages 24–34, Plovdiv, Bulgaria. IEEE.

[Dejaeghere et al. 2023] Dejaeghere, J., Gbadamosi, B., Pulls, T., and Rochet, F. (2023). Comparing security in eBPF and WebAssembly. In *Proceedings of the 1st Workshop on EBPF and Kernel Extensions*, pages 35–41, New York, NY, USA. ACM.

[Disselkoen et al. 2019] Disselkoen, C., Renner, J., Watt, C., Garfinkel, T., Levy, A., and Stefan, D. (2019). Position paper: Progressive memory safety for WebAssembly. In *Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy*, pages 1–8, Phoenix, AZ, USA. ACM.

[Donovan et al. 2010] Donovan, A., Muth, R., Chen, B., and Sehr, D. (2010). PNaCl: Portable native client executables. White paper, Google. https://www.chromium.org/nativeclient/reference/research-papers/pnacl.pdf.

[Durumeric 2023] Durumeric, Z. (2023). Cached chrome top million websites. https://github.com/zakird/crux-top-lists.

[Emscripten 2021] Emscripten (2021). Emscripten is a complete compiler toolchain to WebAssembly, using LLVM, with a special focus on speed, size, and the Web platform. https://emscripten.org/.

[EOSIO 2023] EOSIO (2023). EOSIO is a next-generation, open-source blockchain protocol with industry-leading transaction speed and flexible utility. https://github.com/EOSIO.

[Feldman 2019] Feldman, R. (2019). Why isn't functional programming the norm? https://tinyurl.com/yyhtxt74.

[Felker 2023] Felker, R. (2023). Musl libc. https://musl.libc.org/.

[Fox and Patterson 2021] Fox, A. and Patterson, D. (2021). *Engineering Software as a Service: An Agile Approach Using Cloud Computing*. Version 2.0b7, 2nd edition. https://saasbook.info/.

[Fraiwan et al. 2012] Fraiwan, M., Al-Salman, R., Khasawneh, N., and Conrad, S. (2012). Analysis and identification of malicious JavaScript code. *Information Security Journal: A Global Perspective*, 21(1):1–11.

[fstar-lang 2023] fstar-lang (2023). Introduction to F*. https://fstar-lang.org/.

[Fu et al. 2018] Fu, W., Lin, R., and Inge, D. (2018). TaintAssembly: Taint-based information flow control tracking for WebAssembly. arXiv preprint 1802.01050. http://doi.org/10.48550/arxiv.1802.01050.

[Gadepalli et al. 2020] Gadepalli, P. K., McBride, S., Peach, G., Cherkasova, L., and Parmer, G. (2020). Sledge: A serverless-first, light-weight Wasm runtime for the edge. In *Proceedings of the 21st International Middleware Conference*, pages 265–279, Delft, Netherlands. ACM.

[Gao et al. 2022] Gao, S., Fu, H., Zhang, H., Zhang, J., and Li, G. (2022). ZAWA: A ZKSNARK WASM emulator. *Proceedings of the ACM on Programming Languages*, 1(1).

[Genkin et al. 2018] Genkin, D., Pachmanov, L., Tromer, E., and Yarom, Y. (2018). Drive-by key-extraction cache attacks from portable code. In *Proceedings of the 16th International Conference on Applied Cryptography and Network Security*, pages 83–102, Leuven, Belgium. Springer.

[Golsch 2019] Golsch, L. (2019). WebAssembly: Basics. Technical report, Technical University of Braunschweig. https://www.lennard-golsch.de/article/webassembly.pdf.

[Goltzsche et al. 2019] Goltzsche, D., Nieke, M., Knauth, T., and Kapitza, R. (2019). AccTEE: A WebAssembly-based two-way sandbox for trusted resource accounting. In *Proceedings of the 20th International Middleware Conference*, pages 123–135, Davis, CA, USA. ACM.

[Goltzsche et al. 2020] Goltzsche, D., Siebels, T., Golsch, L., and Kapitza, R. (2020). Hector: Using untrusted browsers to provision Web applications. arXiv 2010.09512. 10.48550/arXiv.2010.09512.

[Grosskurth and Godfrey 2006] Grosskurth, A. and Godfrey, M. W. (2006). Architecture and evolution of the modern web browser. *Preprint submitted to Elsevier Science*, 12(26):235–246.

[Haas et al. 2017] Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., and Bastien, J. (2017). Bringing the Web up to speed with WebAssembly. In *Proceedings of the 38th Conference on Programming Language Design and Implementation*, pages 185–200, Barcelona, Spain. ACM.

[HACL* 2023] HACL* (2023). A High Assurance Cryptographic Library. https://hacl-star.github.io/.

[Haßler and Maier 2021] Haßler, K. and Maier, D. (2021). WAFL: Binary-only WebAssembly fuzzing with fast snapshots. In *Proceedings of the 5th Reversing and Offensive-Oriented Trends Symposium*, pages 23–30, Vienna, Austria. ACM.

[He et al. 2021] He, N., Zhang, R., Wang, H., Wu, L., Luo, X., Guo, Y., Yu, T., and Jiang, X. (2021). EOSAFE: Security analysis of EOSIO smart contracts. In *Proceedings of the 30th USENIX Security Symposium*, pages 1271–1288, Vancouver, BC, Canada. USENIX Association.

[Heinrich et al. 2023] Heinrich, T., Will, N. C., Obelheiro, R. R., and Maziero, C. A. (2023). Uso de chamadas WASI para a identificação de ameaças em aplicações WebAssembly. In *Anais do XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, Juiz de Fora, MG, Brasil. SBC.

[Heinrich et al. 2024] Heinrich, T., Will, N. C., Obelheiro, R. R., and Maziero, C. A. (2024). A categorical data approach for anomaly detection in WebAssembly applications. In *Proceedings of the 10th International Conference on Information Systems Security and Privacy*, pages 275–284, Rome, Italy. SciTePress.

[Helpa et al. 2024] Helpa, C., Heinrich, T., Botacin, M., Will, N. C., Obelheiro, R. R., and Maziero, C. (2024). The use of the DWARF Debugging Format for the Identification of Potentially Unwanted Applications (PUAs) in WebAssembly Binaries. In *SECRYPT*.

[Helpa et al. 2023] Helpa, C., Heinrich, T., Botacin, M., Will, N. C., Obelheiro, R. R., and Maziero, C. A. (2023). Uma estratégia dinâmica para a detecção de anomalias em binários WebAssembly. In *Anais do XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, Juiz de Fora, MG, Brasil. SBC.

[Herman et al. 2014] Herman, D., Wagner, L., and Zakai, A. (2014). asm.js. `http://asmjs.org/spec/latest/`.

[Hilbig et al. 2021] Hilbig, A., Lehmann, D., and Pradel, M. (2021). An empirical study of real-world WebAssembly binaries: Security, languages, use cases. In *Proceedings of The Web Conference*, pages 2696–2708, Ljubljana, Slovenia. ACM.

[Hoffman 2019] Hoffman, K. (2019). *Programming WebAssembly with Rust: unified development for web, mobile, and embedded applications*. The Pragmatic Bookshelf, Raleigh, NC, USA.

[Jangda et al. 2019] Jangda, A., Powers, B., Berger, E. D., and Guha, A. (2019). Not so fast: Analyzing the performance of WebAssembly vs. native code. In *Proceedings of the USENIX Annual Technical Conference*, pages 107–120, Renton, WA, USA. USENIX Association.

[Johnson et al. 2023] Johnson, E., Laufer, E., Zhao, Z., Gohman, D., Narayan, S., Savage, S., Stefan, D., and Brown, F. (2023). WaVe: a verifiably secure WebAssembly sandboxing runtime. In *Proceedings of the Symposium on Security and Privacy*, pages 2940–2955, San Francisco, CA, USA. IEEE.

[Kim et al. 2022] Kim, M., Jang, H., and Shin, Y. (2022). Avengers, Assemble! survey of WebAssembly security solutions. In *Proceedings of the 15th International Conference on Cloud Computing*, pages 543–553, Barcelona, Spain. IEEE.

[Kocher et al. 2020] Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., et al. (2020). Spectre attacks: Exploiting speculative execution. *Communications of the ACM*, 63(7):93–101.

[Kolosick et al. 2022] Kolosick, M., Narayan, S., Johnson, E., Watt, C., LeMay, M., Garg, D., Jhala, R., and Stefan, D. (2022). Isolation without taxation: Near-zero-cost transitions for WebAssembly and SFI. *Proceedings of the ACM on Programming Languages*, 6(POPL).

[Koren 2021] Koren, I. (2021). A standalone WebAssembly development environment for the internet of things. In *Proceedings of the 21st International Conference on Web Engineering*, pages 353–360, Biarritz, France. Springer.

[Le Pochat et al. 2019] Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczyński, M., and Joosen, W. (2019). Tranco: A research-oriented top sites ranking hardened against manipulation. In *26th Annual Network and Distributed System Security Symposium*, NDSS '19. `10.14722/ndss.2019.23386`.

[Lehmann et al. 2020] Lehmann, D., Kinder, J., and Pradel, M. (2020). Everything old is new again: Binary security of WebAssembly. In *Proceedings of the 29th USENIX Security Symposium*, pages 217–234, Virtual Event. USENIX Association.

[Lehmann and Pradel 2019] Lehmann, D. and Pradel, M. (2019). Wasabi: A framework for dynamically analyzing WebAssembly. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1045–1058, Providence, RI, USA. ACM.

[Lehmann and Pradel 2022] Lehmann, D. and Pradel, M. (2022). Finding the dwarf: Recovering precise types from WebAssembly binaries. In *Proceedings of the 43rd International Conference on Programming Language Design and Implementation*, pages 410–425, San Diego, CA, USA. ACM.

[Lehmann et al. 2021] Lehmann, D., Torp, M. T., and Pradel, M. (2021). Fuzzm: Finding memory bugs through binary-only instrumentation and fuzzing of WebAssembly.

[Levick 2024] Levick, R. (2024). Deconstructing webassembly components. `https://www.youtube.com/@wasmio`.

[Li et al. 2021] Li, B., Fan, H., Gao, Y., and Dong, W. (2021). ThingSpire OS: A WebAssembly-based IoT operating system for cloud-edge integration. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 487–488, Virtual Event. ACM.

[Li and Zhang 2022] Li, L. T. and Zhang, M. (2022). Poster: EOSDFA: Data flow analysis of EOSIO smart contracts. In *Proceedings of the Conference on Computer and Communications Security*, pages 3391–3393, Los Angeles, CA, USA. ACM.

[Liang et al. 2018] Liang, H., Pei, X., Jia, X., Shen, W., and Zhang, J. (2018). Fuzzing: State of the art. *IEEE Transactions on Reliability*, 67(3):1199–1218.

[Liu et al. 2021] Liu, R., Garcia, L., and Srivastava, M. (2021). Aerogel: Lightweight access control framework for WebAssembly-based bare-metal IoT devices. In *Proceedings of the 6th Symposium on Edge Computing*, pages 94–105, San Jose, CA, USA. IEEE.

[LLVM 2023] LLVM (2023). The LLVM compiler infrastructure. `https://github.com/llvm/llvm-project`.

[LLVM-Team 2023] LLVM-Team (2023). The LLVM project. `https://llvm.org/`.

[Marques et al. 2022] Marques, F., Fragoso Santos, J., Santos, N., and Adão, P. (2022). Concolic execution for WebAssembly. In *Proceedings of the 36th European Conference on Object-Oriented Programming*, Berlin, Germany. DROPS.

[Mazaheri et al. 2022] Mazaheri, M. E., Bayat Sarmadi, S., and Taheri Ardakani, F. (2022). A study of timing side-channel attacks and countermeasures on JavaScript and WebAssembly. *The ISC International Journal of Information Security*, 14(1):27–46.

[McFadden et al. 2018] McFadden, B., Lukasiewicz, T., Dileo, J., and Engler, J. (2018). Security chasms of WASM. *NCC Group Whitepaper*.

[Ménétrey et al. 2021] Ménétrey, J., Pasin, M., Felber, P., and Schiavoni, V. (2021). Twine: An embedded trusted runtime for WebAssembly. In *Proceedings of the 37th International Conference on Data Engineering*, pages 205–216, Chania, Greece. IEEE.

[Ménétrey et al. 2022] Ménétrey, J., Pasin, M., Felber, P., and Schiavoni, V. (2022). WebAssembly as a common layer for the cloud-edge continuum. In *Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge*, pages 3–8, Minneapolis, MN, USA. ACM.

[Michael et al. 2023] Michael, A. E., Gollamudi, A., Bosamiya, J., Disselkoen, C., Denlinger, A., Watt, C., Parno, B., Patrignani, M., Vassena, M., and Stefan, D. (2023). MSWasm: Soundly enforcing memory-safe execution of unsafe code. *Proceedings of the ACM on Programming Languages*, 1(1).

[Microsoft 1996] Microsoft (1996). Microsoft announces activex technologies. `https://news.microsoft.com/1996/03/12/microsoft-announces-activex-technologies/`.

[Ming et al. 2016] Ming, J., Wu, D., Wang, J., Xiao, G., and Liu, P. (2016). StraightTaint: Decoupled offline symbolic taint analysis. In *Proceedings of the 31st International Conference on Automated Software Engineering*, pages 308–319, Singapore. IEEE.

[Munsters et al. 2021] Munsters, A., Pupo, A. L. S., Bauwens, J., and Boix, E. G. (2021). Oron: Towards a dynamic analysis instrumentation platform for AssemblyScript. In *Proceedings of the 5th International Conference on the Art, Science, and Engineering of Programming*, pages 6–13, Cambridge, UK. ACM.

[Musch et al. 2019] Musch, M., Wressnegger, C., Johns, M., and Rieck, K. (2019). New kid on the web: A study on the prevalence of webassembly in the wild. In *Proceedings of the 16th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 23–42, Gothenburg, Sweden. Springer.

[Ménétrey et al. 2022] Ménétrey, J., Pasin, M., Felber, P., and Schiavoni, V. (2022). WaTZ: A trusted WebAssembly runtime environment with remote attestation for TrustZone. In *Proceedings of the 42nd International Conference on Distributed Computing Systems*, pages 1177–1189, Bologna, Italy. IEEE.

[Narayan et al. 2020] Narayan, S., Disselkoen, C., Garfinkel, T., Froyd, N., Rahm, E., Lerner, S., Shacham, H., and Stefan, D. (2020). Retrofitting fine grain isolation in the Firefox renderer. In *Proceedings of the 29th USENIX Security Symposium*, pages 699–716, Boston, MA, USA. USENIX Association.

[Narayan et al. 2021] Narayan, S., Disselkoen, C., Moghimi, D., Cauligi, S., Johnson, E., Gang, Z., Vahldiek-Oberwagner, A., Sahita, R., Shacham, H., Tullsen, D., et al. (2021). Swivel: Hardening WebAssembly against Spectre. In *Proceedings of the 30th USENIX Security Symposium*, pages 1433–1450, Vancouver, BC, Canada. USENIX Association.

[Narayan et al. 2019] Narayan, S., Garfinkel, T., Lerner, S., Shacham, H., and Stefan, D. (2019). Gobi: WebAssembly as a practical path to library sandboxing. arXiv preprint 1912.02285. 10.48550/arXiv.1912.02285.

[Nieke et al. 2021] Nieke, M., Almstedt, L., and Kapitza, R. (2021). Edgedancer: Secure mobile WebAssembly services on the edge. In *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, pages 13–18, Virtual Event. ACM.

[Petrov et al. 2020] Petrov, I., Invernizzi, L., and Bursztein, E. (2020). CoinPolice: Detecting hidden cryptojacking attacks with neural networks. arXiv preprint 2006.10861. 10.48550/arXiv.2006.10861.

[Pop et al. 2022] Pop, V. A. B., Niemi, A., Manea, V., Rusanen, A., and Ekberg, J.-E. (2022). Towards securely migrating WebAssembly enclaves. In *Proceedings of the 15th European Workshop on Systems Security*, pages 43–49, Rennes, France. ACM.

[Protzenko et al. 2019] Protzenko, J., Beurdouche, B., Merigoux, D., and Bhargavan, K. (2019). Formally verified cryptographic Web applications in WebAssembly. In *Proceedings of the 40th Symposium on Security and Privacy*, pages 1256–1274, San Francisco, CA, USA. IEEE.

[Qiang et al. 2018] Qiang, W., Dong, Z., and Jin, H. (2018). Se-Lambda: Securing privacy-sensitive serverless applications using SGX enclave. In *Proceedings of the 14th International Conference on Security and Privacy in Communication Systems*, pages 451–470, Singapore. Springer.

[Quan et al. 2019] Quan, L., Wu, L., and Wang, H. (2019). EVulHunter: Detecting fake transfer vulnerabilities for EOSIO's smart contracts at Webassembly-level. 10.48550/arXiv.1906.10362.

[Radovici et al. 2018] Radovici, A., Rusu, C., and Serban, R. (2018). A survey of IoT security threats and solutions. In *Proceedings of the 17th RoEduNet Conference: Networking in Education and Research*, pages 1–5, Cluj-Napoca, Romania. IEEE.

[Renner et al. 2018] Renner, J., Cauligi, S., and Stefan, D. (2018). Constant-time WebAssembly. In *Proceedings of the 45th Symposium on Principles of Programming Languages*, Los Angeles, CA, USA. ACM.

[Rokicki 2022] Rokicki, T. (2022). *Side Channels in Web Browsers: Applications to Security and Privacy*. Thèse de doctorat, Institut National des Sciences Appliquées de Rennes. https://hal.science/tel-04493651/.

[Romano et al. 2022] Romano, A., Lehmann, D., Pradel, M., and Wang, W. (2022). Wobfuscator: Obfuscating JavaScript malware via opportunistic translation to WebAssembly. In *Proceedings of the 43rd Symposium on Security and Privacy*, pages 1574–1589, San Francisco, CA, USA. IEEE.

[Romano et al. 2021] Romano, A., Liu, X., Kwon, Y., and Wang, W. (2021). An empirical study of bugs in WebAssembly compilers. In *Proceedings of the 36th International Conference on Automated Software Engineering*, pages 42–54, Melbourne, Australia. IEEE.

[Romano and Wang 2020a] Romano, A. and Wang, W. (2020a). Wasim: Understanding WebAssembly applications through classification. In *Proceedings of the 35th International Conference on Automated Software Engineering*, pages 1321–1325, Melbourne, Australia. IEEE.

[Romano and Wang 2020b] Romano, A. and Wang, W. (2020b). WasmView: Visual testing for WebAssembly applications. In *Proceedings of the 42nd International Conference on Software Engineering*, pages 13–16, Seoul, South Korea. ACM.

[Rossberg 2024] Rossberg, A. (2024). WebAssembly specification. `https://webassembly.github.io/spec/core/index.html`.

[Rourke 2018] Rourke, M. (2018). *Learn WebAssembly: Build web applications with native performance using Wasm and C/C++*. Packt Publishing Ltd.

[Ruth et al. 2022] Ruth, K., Kumar, D., Wang, B., Valenta, L., and Durumeric, Z. (2022). Toppling top lists: Evaluating the accuracy of popular website lists. In *Proceedings of the 22nd Internet Measurement Conference*, pages 374–387, Nice, France. ACM.

[Shirey 2007] Shirey, R. (2007). Internet security glossary, version 2. RFC 4949. `https://www.rfc-editor.org/info/rfc4949`.

[Sletten 2022] Sletten, B. (2022). *WebAssembly: The Definitive Guide*. O'Reilly Media, Inc.

[Spreitzer et al. 2017] Spreitzer, R., Moonsamy, V., Korak, T., and Mangard, S. (2017). Systematic classification of side-channel attacks: A case study for mobile devices. *IEEE Communications Surveys & Tutorials*, 20(1):465–488.

[stackoverflow 2023] stackoverflow (2023). Stackoverflow. `https://stackoverflow.com/`.

[Stephen 2022] Stephen, A. (2022). Awesome WebAssembly languages. `https://github.com/appcypher/awesome-wasm-langs`.

[Stiévenart et al. 2022a] Stiévenart, Q., Binkley, D. W., and De Roover, C. (2022a). Static stack-preserving intra-procedural slicing of WebAssembly binaries. In *Proceedings of the 44th International Conference on Software Engineering*, pages 2031–2042, Pittsburgh, PA, USA. IEEE.

[Stiévenart and De Roover 2020] Stiévenart, Q. and De Roover, C. (2020). Compositional information flow analysis for WebAssembly programs. In *Proceedings of the 20th International Working Conference on Source Code Analysis and Manipulation*, pages 13–24, Adelaide, Australia. IEEE.

[Stiévenart et al. 2021] Stiévenart, Q., De Roover, C., and Ghafari, M. (2021). The security risk of lacking compiler protection in WebAssembly. In *Proceedings of the 21st International Conference on Software Quality, Reliability and Security*, pages 132–139, Hainan, China. IEEE.

[Stiévenart et al. 2022b] Stiévenart, Q., De Roover, C., and Ghafari, M. (2022b). Security risks of porting C programs to WebAssembly. In *Proceedings of the 37th Symposium on Applied Computing*, pages 1713–1722, Virtual Event. ACM.

[Stock et al. 2017] Stock, B., Johns, M., Steffens, M., and Backes, M. (2017). How the Web tangled itself: Uncovering the history of client-side Web (in)security. In *Proceedings of the 26th USENIX Security Symposium*, pages 971–987, Vancouver, BC, Canada. USENIX Association.

[Sun et al. 2019] Sun, J., Cao, D., Liu, X., Zhao, Z., Wang, W., Gong, X., and Zhang, J. (2019). SELWasm: A code protection mechanism for WebAssembly. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing with Applications*, pages 1099–1106, Xiamen, China. IEEE.

[Sun et al. 2022] Sun, S., Ma, H., Song, Z., and Zhang, R. (2022). WebCloud: Web-based cloud storage for secure data sharing across platforms. *IEEE Transactions on Dependable and Secure Computing*, 19(3):1871–1884.

[Szanto et al. 2018] Szanto, A., Tamm, T., and Pagnoni, A. (2018). Taint tracking for WebAssembly. *arXiv preprint arXiv:1807.08349*.

[Titzer 2022] Titzer, B. L. (2022). A fast in-place interpreter for WebAssembly. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2).

[Tiwari et al. 2018] Tiwari, T., Starobinski, D., and Trachtenberg, A. (2018). Distributed Web mining of Ethereum. In *Proceedings of the International Symposium on Cyber Security Cryptography and Machine Learning*, pages 38–54, Beer-Sheva, Israel. Springer.

[Tsoupidi et al. 2021] Tsoupidi, R. M., Balliu, M., and Baudry, B. (2021). Vivienne: Relational verification of cryptographic implementations in WebAssembly. In *Proceedings of the Secure Development Conference*, pages 94–102, Atlanta, GA, USA. IEEE.

[Vassena et al. 2021] Vassena, M., Disselkoen, C., Gleissenthall, K. v., Cauligi, S., Kıcı, R. G., Jhala, R., Tullsen, D., and Stefan, D. (2021). Automatically eliminating speculative leaks from cryptographic code with Blade. *Proceedings of the ACM on Programming Languages*, 5(POPL).

[Vassena and Patrignani 2020] Vassena, M. and Patrignani, M. (2020). Memory safety preservation for WebAssembly. In *Proceedings of the 47th Symposium on Principles of Programming Languages*, New Orleans, LA, USA. ACM.

[W3C Community Group 2022a] W3C Community Group (2022a). WA: Security. `https://webassembly.org/docs/security/`.

[W3C Community Group 2022b] W3C Community Group (2022b). WebAssembly. https://webassembly.org.

[W3Techs 2023] W3Techs (2023). Usage statistics of JavaScript as client-side programming language on websites. https://w3techs.com/technologies/details/cp-javascript.

[Wang et al. 2019] Wang, S., Ye, G., Li, M., Yuan, L., Tang, Z., Wang, H., Wang, W., Wang, F., Ren, J., Fang, D., and Wang, Z. (2019). Leveraging WebAssembly for numerical JavaScript code virtualization. *IEEE Access*, 7:182711–182724.

[Wang 2021] Wang, W. (2021). Empowering web applications with WebAssembly: are we there yet? In *Proceedings of the 36th International Conference on Automated Software Engineering*, pages 1301–1305, Melbourne, Australia. IEEE.

[Wang et al. 2018] Wang, W., Ferrell, B., Xu, X., Hamlen, K. W., and Hao, S. (2018). SEISMIC: SEcure In-lined Script Monitors for Interrupting Cryptojacks. In *Proceedings of the 23rd European Symposium on Research in Computer Security*, pages 122–142, Barcelona, Spain. Springer.

[Watt et al. 2019a] Watt, C., Renner, J., Popescu, N., Cauligi, S., and Stefan, D. (2019a). CT-Wasm: Type-driven secure cryptography for the Web ecosystem. *Proceedings of the ACM on Programming Languages*, 3(POPL).

[Watt et al. 2019b] Watt, C., Rossberg, A., and Pichon-Pharabod, J. (2019b). Weakening WebAssembly. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–28.

[WebAssembly 2024] WebAssembly (2024). Webassembly proposals. https://github.com/WebAssembly/proposals.

[Wen and Weber 2020] Wen, E. and Weber, G. (2020). Wasmachine: Bring the edge up to speed with a WebAssembly OS. In *Proceedings of the 13th International Conference on Cloud Computing*, pages 353–360, Beijing, China. IEEE.

[Wen et al. 2021] Wen, E., Weber, G., and Nanayakkara, S. (2021). WasmAndroid: A cross-platform runtime for native programming languages on Android (WIP paper). In *Proceedings of the 22nd International Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 80–84, Virtual Event. ACM.

[Will et al. 2021] Will, N. C., Heinrich, T., Viescinski, A. B., and Maziero, C. A. (2021). Trusted inter-process communication using hardware enclaves. In *Proceedings of the 15th International Systems Conference*, pages 1–7, Vancouver, BC, Canada. IEEE.

[Xie et al. 2022] Xie, Q., Tang, S., Zheng, X., Lin, Q., Liu, B., Duan, H., and Li, F. (2022). Building an open, robust, and stable voting-based domain top list. In *Proceedings of the 31st USENIX Security Symposium*, pages 625–642, Boston, MA, USA. USENIX Association.

[Yan et al. 2021] Yan, Y., Tu, T., Zhao, L., Zhou, Y., and Wang, W. (2021). Understanding the performance of WebAssembly applications. In *Proceedings of the 21st Internet Measurement Conference*, pages 533–549, Virtual Event. ACM.

[Yee et al. 2010] Yee, B., Sehr, D., Dardyk, G., Chen, J. B., Muth, R., Ormandy, T., Okasaka, S., Narula, N., and Fullagar, N. (2010). Native Client: A sandbox for portable, untrusted x86 native code. *Communications of the ACM*, 53(1):91–99.

[Yu et al. 2020] Yu, G., Yang, G., Li, T., Han, X., Guan, S., Zhang, J., and Gu, G. (2020). MinerGate: A novel generic and accurate defense solution against Web based cryptocurrency mining attacks. In *Proceedings of the 17th China Cyber Security Annual Conference*, pages 50–70, Beijing, China. Springer.

[Zakai 2011] Zakai, A. (2011). Emscripten: An LLVM-to-JavaScript compiler. In *Proceedings of the International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, pages 301–312, Portland, Oregon, USA. ACM.

[Zhang et al. 2024] Zhang, Y., Zheng, S., Wang, H., Wu, L., Huang, G., and Liu, X. (2024). VM matters: A comparison of WASM VMs and EVMs in the performance of blockchain smart contracts. *ACM Transactions on Modeling and Performance Evaluation of Computer Systems*, 9(2).

SBRC'24

42º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos

20 A 24
MAIO
NITERÓI

https://sbrc.sbc.org.br/2024/

Realizado por

Apoiado por

Patrocinado por