electronics

Article

Implementation of a Lossless Moving Target Defense Mechanism

Mariusz Żal , Marek Michalski and Piotr Zwierzykowski *

Institute of Communication and Computer Networks, Faculty of Computing and Telecommunications, Poznań University of Technology, 60-965 Poznań, Poland; mariusz.zal@put.poznan.pl (M.Ż); marek.michalski@put.poznan.pl (M.M.)
* Correspondence: piotr.zwierzykowski@put.poznan.pl; Tel.: +48-61-665-39-26

Abstract: The contemporary world, dominated by information technologt (IT), necessitates sophisticated protection mechanisms against attacks that pose significant threats to individuals, companies, and governments alike. The unpredictability of human behavior, coupled with the scattered development of applications and devices, complicates supply chain maintenance, making it impossible to develop a system entirely immune to cyberattacks. Effective execution of many attack types hinges on prior network reconnaissance. Thus, hindering effective reconnaissance serves as a countermeasure to attacks. This paper introduces a solution within the moving target defense (MTD) strategies, focusing on the mutation of Internet protocol (IP) addresses in both edge and core network switches. The idea of complicating reconnaissance by continually changing IP addresses has been suggested in numerous studies. Nonetheless, previously proposed solutions have adversely impacted the quality of service (QoS) levels. Implementing these mechanisms could interrupt Transmission Control Protocol (TCP) connections and result in data losses. The IP address mutation algorithms presented in this study were designed to be fully transparent to transport layer protocols, thereby preserving the QoS for users without degradation. In this study, we leveraged the benefits of software-defined networking (SDN) and the Programming-Protocol-Ondependent Packet Processors (P4) language, which specifies packet processing methodologies in the data plane. Employing both SDN and P4 enables a dynamic customization of network device functionalities to meet network users' specific requirements, a feat unachievable with conventional computer networks. This approach not only enhances the adaptability of network configurations but also significantly increases the efficiency and effectiveness of network management and operation.

Keywords: moving target defense; software-defined networks; Programming-Protocol-Independent Packet Processors (P4) language; IP address mutation; cybersecurity

check for updates

Citation: Żal, M.; Michalski, M.; Zwierzykowski, P. Implementation of a Lossless Moving Target Defense Mechanism. *Electronics* **2024**, *13*, 918. https://doi.org/10.3390/electronics13050918

Academic Editor: Christos J. Bouras

Received: 31 December 2023
Revised: 21 February 2024
Accepted: 26 February 2024
Published: 28 February 2024

1. Introduction

A criminal planning to intentionally break the law does not manifest their intentions. The aim is to remain unnoticed by those in their surroundings for as long as possible, especially by security institutions such as the police, intelligence, and counterintelligence services. The main operational and reconnaissance method of state services is the observation of people, places, and things; therefore, from the perspective of a criminal, and often a criminal group, it is completely justified to carry out counter-intelligence activities [1]. Criminals carefully plan their next moves. They do everything possible to ensure that the process of gathering information, which could aid in committing a prohibited act, is unnoticed. They meticulously cover up traces that could indicate who committed the crime. If the crime is ongoing or meant to last for a longer period, the criminal disguises their presence and the manner in which the crime was committed. Describing the criminal mechanism is not a praise for the level of intelligence and cunning of criminals; it only indicates the costs incurred during the commission of crimes. For the commission of a crime to be profitable for the criminal (taking into account the costs of preparation and the probability of punishment), the value of the object of the crime must be significantly higher than the costs.

Cybercriminals may also follow a pattern. The process of gathering information about a system is called reconnaissance [2,3]. Paradoxically, reconnaissance was originally used to gather information about systems in order to identify security vulnerabilities. Originally, it was an ethical hacking technique that allowed network owners to better secure their systems after identifying security gaps. It should be noted that not all hackers are criminals. The term "hacker" is frequently misapplied to individuals engaging in malicious security breaches for personal gain, often with criminal intentions. In hacker communities, such people are called crackers. Over the years, reconnaissance has evolved from an ethical hacking procedure to a mechanism of cyber attack. A reconnaissance attack is a process in which the hacker takes on the role of a secret detective to obtain information about target systems. This information is then used to identify security vulnerabilities before launching an attack or to pinpoint resources that may be targeted.

Reconnaissance attacks are most often carried out from within the targeted system. We distinguish between active and passive reconnaissance [4]. In active reconnaissance, the attacker interacts directly with the target. This can take place on multiple levels. To obtain confidential information, the attacker may use social engineering, such as sending emails, using chatbots, or other interactive communication means, to establish a connection. Another method is port scanning, which entails checking open (active) Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) protocol ports by sending data transfer requests to the tested ports. Active footprinting is also a technique used, involving actions aimed at checking active Internet Protocol (IP) addresses or email addresses, by interacting with the relevant services. Active reconnaissance is extremely efficient, as it is targeted at obtaining data essential for conducting an attack and is relatively quick. A notable defense feature against such attacks is the possibility of detection, since the attacker's interactions with the system can be identified as deviations from normal activity. The second type of reconnaissance is passive reconnaissance. In this approach, the attacker does not directly interact with the user or their system. They conduct their investigation remotely, monitoring traffic and interactions on the network. The attacker can collect and analyze data from public resources, in a process known as open source intelligence (OSINT) [5]. Both individuals and networks disseminate their information on the network, whether intentionally or unintentionally. A person conducting reconnaissance can utilize Open Source Intelligence (OSINT) to obtain valuable information about a system.

Just as there is no human activity or action that is not in some way fraught with some risk that may affect health or life, no network exists that has not been or will not be the target of an attack. The attacker has many ways to "lurk" within the system, including using social engineering, exploiting breaks in the supply chain, or deploying zero-day attacks. Therefore, it is necessary to take actions aimed at making reconnaissance difficult. One possible method involves disorienting the attacker by indicating that the collected data are worthless, which means the process of collection and analysis must start from scratch. This could result in an increased cost of the attack, potentially rendering it unprofitable. This also increases the chances of detecting the attacker. One such method is Moving Target Defense (MTD), in which dynamic changes are made to the parameters of the protected system that enable its identification or determine its structure [6]. A significant drawback of the method is that involves changes made to an operating system, which can affect the continuity of provided services. This paper proposes a method for the dynamic mutation of IP addresses of hosts operating in a protected network. IP address mutation is not a new technique for hindering reconnaissance. A drawback of the published solutions is that they do not account for the transitional state in which network IP addressing is inconsistent. This oversight can cause disruptions in the service at the transport layer level, interrupting established TCP connections. This can lead to data loss, thereby degrading the level of Quality of Service (QoS) [7–9]. The presented IP mutation mechanisms in many implementations utilize the Dynamic Host Configuration Protocol (DHCP) to change IP addresses. In such a scenario, simultaneous IP address changes by all devices operating in the network are not possible. Moreover, this may require introducing changes at the operating system level to adjust the operation of network applications to the IP address changes. The aim of this study was to develop a solution utilizing IP address mutation that does not affect the functioning of transport-layer protocols. Since IP address mutation is completely transparent to protocols using IP protocol services, it avoids connection interruption or data loss. In the proposed solution, Software-Defined Network (SDN) is utilized [10–12]. However, the characteristics of SDN networks alone do not address the issue of of maintaining TCP connections. In the proposed solution, the Protocol-Independent Packet Processor Programming (P4) language and the Portable Switch Architecture (PSA) are employed, which offer another level of freedom in preparing network applications. The ability to implement packet processing algorithms directly in the data path helps to resolve the issue of temporary inconsistency in IP addressing.

This article is organized as follows: The next section contains a literature review on MTD techniques. Section 3 introduces the concept of SDNs. Since every network equipment provider currently introduces their own solutions that fall within the SDN domain, the we describe the most crucial elements of SDNs used in implementation. The next section provides a description of the individual stages of the cyberattack process and how to defend against it, introducing the ideas behind the MTD technique. Section 5 contains a description of the P4 language, focusing on presenting the most important elements of the language essential for understanding the operation of the algorithm. In the following section, an evaluation of the performance of the MTD mechanism using IP address mutation is performed. The network convergence time was determined, i.e., the time after which all switches use only mutated IP addresses belonging to the same group, referred to as a generation. We also present formulas specifying how often addresses can be mutated. The next section provides a detailed description of the proposed MTD mechanism and its implementation using a simple model (V1Model) of the P4 switch. This article concludes with conclusions and plans for further research work.

2. State of the Art

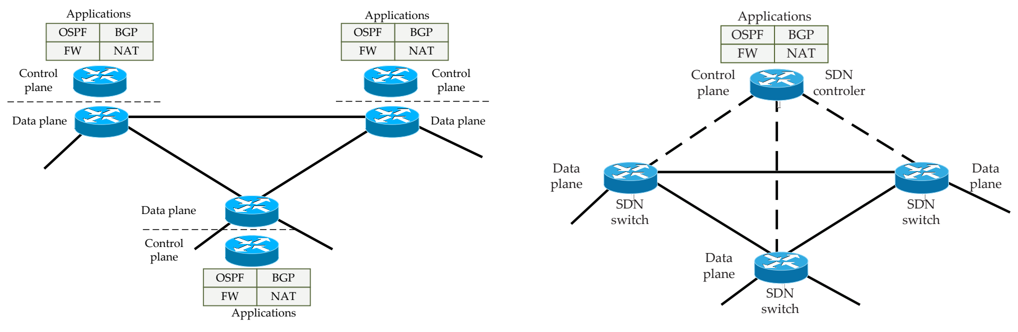
The evolution of networks and computer systems has also forced the evolution of cyberattacks. Cyberattacks that were used 10 or 20 years ago are now considered simple or even primitive. The attacks used currently can be classified as smart attacks. Modern attacks, which use innovative approaches, are difficult to detect and resistant to traditional methods of defense. The reputation that an attacker could achieve in their environment has ceased to be a sufficient form of gratification. Attacks have become a source of livelihood for whole groups of people. Often, their execution is supported by funds from various organizations or governments that are considered to support terrorism. Such attacks have become very complicated and are adapted to the changing conditions of the attacked area. Only intelligent defense mechanisms can counteract intelligent attacks. Among such mechanisms, we include the MTD technique.

MTD can be applied at many levels: data, application, runtime environment, operating system, or hardware. Moving target techniques in the dynamic data domain change the format, syntax, representation, or encoding of application data to complicate attacks. An interesting use of dynamic data techniques is data space randomization (DSR), where either the data space or the program code is modified [13]. Another example of an MTD technique, which falls within the category of dynamic applications, involves an environment utilizing several web application servers that perform the same functions. The MTD mechanisms used in this solution dynamically redirect client requests to any chosen server [14]. This mitigates the effects of vulnerabilities in server software. A product by Morphisec Labs exemplifies a dynamic runtime environment, a technique where the execution environment, including RAM addresses and instruction sets, changes dynamically [15]. In computer systems, the operating system, memory, and processor, with its instruction set being closely related, are interconnected. Therefore, MTD techniques covering these three elements are treated as a whole and are collectively referred to as a dynamic platform. Examples of the solutions in this area can be found in [16–18].

The most explored research area in the field of MTD is dynamic networks. The techniques proposed in this area suggest modifications to a wide range of parameters. The proposed solutions include dynamic modifications to network topology. The initiation can be triggered randomly [19,20] or at predefined times [21]. Another way to hinder reconnaissance is by changing the paths along which data are exchanged between two hosts [22–26]. In addition to changing the routes between two hosts, modifications to computer networks can also dynamically alter routing protocol data [27,28]. As a result, changes in the network structure are more extensive. Beyond affecting switches and routers, it is also possible to implement changes that exclusively involve hosts. For example, random or timer-triggered changes in port numbers in established connections are possible [29–31].The method most commonly representation in the literature is IP address mutations [12,22,26,32–37]. As with other methods, the change in IP addresses can be initiated at random moments, at times determined by an algorithm (e.g., from game theory [38]), or by detecting anomalies indicating third-party activity. To detect anomalies, specially prepared devices, so-called honeypots [9,39], can be used.

3. SDN Concept

In traditional computer networks, each network node, such as a switch or router, has a fully implemented data plane and a control plane. The proper functioning of such a network necessitates applications running on individual nodes be compatible with each other. Considering that computer networks comprise heterogeneous devices, i.e., from different manufacturers and running different operating systems, application interoperability is sometimes significantly hindered. Furthermore, the need to ensure cooperation between applications restricts the functionality of the network being built. To make a particular service available throughout the network, control-layer applications must be running on all network devices (see Figure 1a). Naturally, applications that perform basic network functions, such as handling routing protocols, i.e., Open Shortest Path First (OSPF), Intermediate System to Intermediate System (IS-IS), Border Gateway Protocol (BGP), etc., or the DHCP protocol, are typically integrated in the operating systems of network devices. The use of advanced functions or applications necessitates their incorporation into the software stack of network devices, which is often an extremely difficult task.



(**a**) Traditional network (**b**) SDN

Figure 1. Differences between traditional networks and SDNs.

In SDNs, device planes are separated, with only the data plane implemented within the devices, and the control plane is separate, as shown in Figure 1b. Devices that perform the tasks of the data plane, known as SDN switches, are responsible for forwarding data between their inputs and outputs. A set of SDN switches managed by the same SDN controller is called an SDN. The rules for forwarding data are defined by devices that perform control plane functions, known as SDN controllers. Information about the rules governing the operation of SDN switches is stored in flow tables. Data exchange between the controller and SDN switches can occur using the standard OpenFlow protocol. The primary advantage of SDN networks is their configurational flexibility and the ease of adding new services and enabling virtualization. This is made possible by using an SDN controller as a central device. In many solutions, centralizing control functions can become a bottleneck for network performance. However, in SDNs, centralizing the control layer along with flow tables on each device implementing the data plane increases network efficiency, allowing for the customization of the features offered to meet user requirements.

The structure of SDNs is defined by the Open Networking Foundation (ONF) [40]. In Figure 2, the basic elements of an SDN are presented. The architecture and elements defined by the ONF should be considered as recommendations only. Each network equipment provider may employ a different approach and their own solutions when it comes to implementing SDNs. The vast majority of network hardware providers utilize the ONF recommendations, in which each SDN controller implements at least two types of interfaces, as shown in Figure 2. The northbound interface is used for communication between the SDN controller and user applications. By using Application Programming Interfaces (APIs) software, applications can influence the operation of the entire network by implementing various scenarios. The availability of a wide range of SDN controllers written in various programming languages allows for the selection of a solution tailored to the specific requirements. The ability to add applications that implement nonstandard functions is a significant advantage of SDNs. Another key component is the southbound interface, which facilitates the exchange of information between the SDN controller and the SDN switch. Through this interface, the controller receives telemetry data from the attached SDN switches and sends flow table updates.

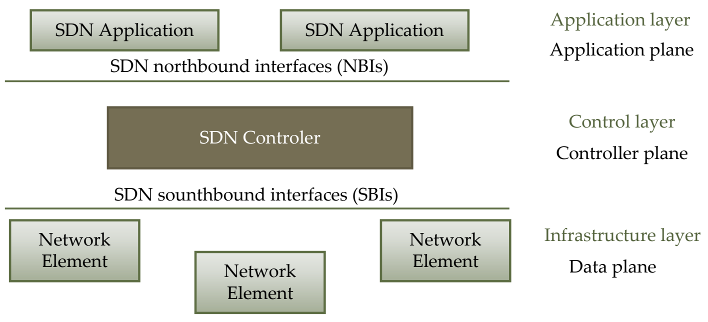
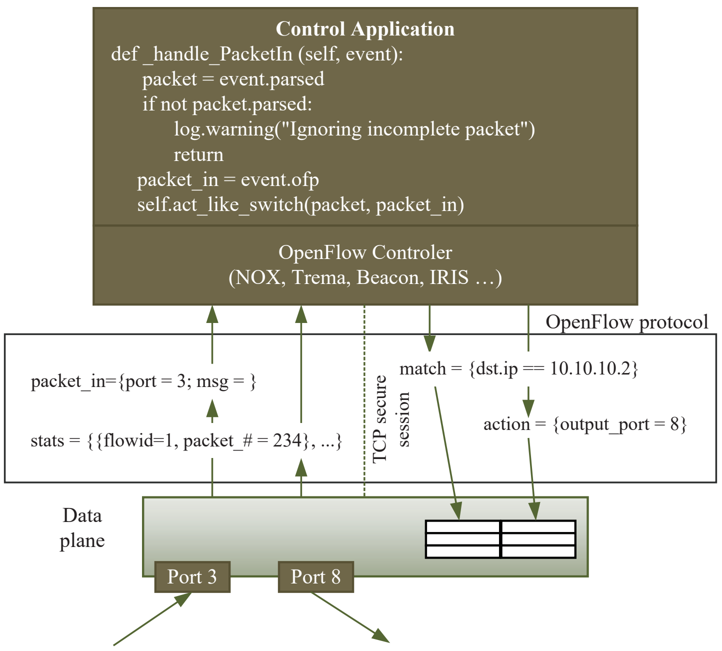


Figure 2. SDN architecture.

The well-known protocol used for data exchange through the southbound interface is the OpenFlow protocol, as described in the standard [41]. Its purpose is to modify the flow table in the controller. As shown in Figure 3, the SDN switch implements a table that consists of two parts: a match part and an action part. From each packet that arrives at the switch's input, data are extracted from the headers of the protocols. These can include the source and destination Media Access Control (MAC) addresses, source and destination IP addresses, protocol type, source, and destination TCP ports. The data are used to look up the appropriate entry in the flow table. In the second part of the flow table, there is information about what action (forwarding to a specific port, removal, modification) needs to be taken on the received packet. The data are generated by the corresponding SDN application. The goal of the OpenFlow protocol is to deliver the data to the SDN switch in a secure and reliable manner. If necessary, the OpenFlow protocol can remove or modify entries in the flow table. In the case of packets for which no matching entry is found in the flow table, the OpenFlow protocol delivers the packet or its fragment to the SDN controller. The controller can then send it to the appropriate SDN application to determine the rules to be added to the flow table. Subsequent packets are then handled according to the updated contents of the flow table. The final task of the OpenFlow protocol is to provide telemetry information from the SDN switch to the relevant SDN applications.



```
def _handle_PacketIn (self, event):
    packet = event.parsed
    if not packet.parsed:
        log.warning("Ignoring incomplete packet")
        return
    packet_in = event.ofp
    self.act_like_switch(packet, packet_in)
```

Control Application
OpenFlow Controler
(NOX, Trema, Beacon, IRIS …)
OpenFlow protocol
packet_in={port = 3; msg = }
stats = {{flowid=1, packet_# = 234}, ...}
TCP secure session
match = {dst.ip == 10.10.10.2}
action = {output_port = 8}
Data plane
Port 3
Port 8

Figure 3. Operation of OpenFlow protocol.

In many cases, the use of a central element is considered a disadvantage because it can become a bottleneck in transmission. In the past, decentralized and distributed control was thought to offer higher efficiency. However, the emergence of SDNs has prompted a re-evaluation of the usefulness of centralized control. A wide range of applications that can participate in preparing the data placed in flow tables, a view of the entire controlled network, and the ability to achieve routing convergence almost instantly are just some of the advantages of SDNs. Another highly important feature of SDNs is the processing control information in only one network element. In the case of routing protocols, e.g., OSPF and IS-IS, or ethernet redundancy protocols like Spanning Tree Protocol (STP) each network node independently creates a view of the network. This leads to each node processing the same data. Utilizing a central element that performs the necessary calculations only once not only reduces electricity consumption but also places SDNs in the category of energy-aware networks.

4. Moving Target Defense

To protect networks from cyberattacks, administrators utilize well-prepared security policies and implement best practices for network maintenance and configuration. They have access to a wide range of advanced tools and procedures, such as software updates to patch vulnerabilities and event log analysis to detect attack-related events or network anomalies. Unfortunately, these techniques are also well known to network attackers.

Between the discovery of a software vulnerability and the release of an update, several days typically pass, and updates are typically performed when the system requiring the update is not in use. This delay provides a significant window of opportunity for carrying out zero-day attacks. Event log analysis may become ineffective when an attack is executed using custom malicious software, which can be challenging to detect or prevent with intrusion detection systems and antivirus tools.

To prepare an effective defense against cyberattacks, the attack process used by adversaries must be understood, and the attack process from the attacker's perspective must be analyzed. Executing a sophisticated and impactful attack on a major network operator, financial organization, government institution, etc., requires substantial financial investments. Attacks on such entities are carried out by Advanced Persistent Threat (APT) groups, sponsored by wealthy organizations or states, driven by economic or political motives. An APT gains access to a computer network and remains undetected for an extended period.

To counteract such activities, understanding what data can be collected by the APT with access to the network is crucial. Identifying the categories of data accessible to the attacking entity enables the pinpointing of potential attack types.

To better understand the entire attack process, the Cyber Kill Chain (CKC) defined in [42] can be used. It consists of the following stages:

1. Reconnaissance—This phase involves gathering data about the environment where the attack will take place. The monitoring area can be extensive, including network parameters such as used protocols and their versions, IP addresses, port numbers, load information, types and versions of applications and operating systems, and the types of services being used or provided.
2. Weaponization—Based on the data obtained in the first stage, the attacker employs various tools and techniques to prepare a payload for a targeted attack. This takes the form of a phishing email, an infected document, or even a modification of a delivered update (by tampering with the supply chain).
3. Delivery—The payload prepared in the previous step must be delivered to the targeted system to initiate infection. The attacker often leverages human factors, typically involving employees of the company or institution, to bypass authentication procedures. Another method of payload delivery may involve exploiting system vulnerabilities or compromising the supply chain.
4. Exploitation—This stage involves executing malicious code, rendering the infected system accessible to the attacker. The attacker gains increased privileges, typically through exploiting a known vulnerability or a zero-day exploit
5. Installation—With increased privileges, the attacker can install malicious software on the victim's computer or begin gathering information from the victim's databases for further actions.
6. Command and Control—This stage includes actions aimed at maintaining remote control over the victim's machine.
7. Actions on Objectives—This is the final stage of the attack, involving actions related to achieving the attacker's objectives. These actions may include downloading critical data, disrupting services, or using the victim's system to conduct further attacks, this time from a trusted system.

MTD is a cybersecurity strategy classified as a dynamic strategy, aimed at actively protecting computer systems, networks, and data by continuously changing the parameters that constitute the so-called attack surface. The attack surface is the space that attackers must explore or reconstruct to determine the configuration of the target system before initiating the actual attack. It comprises all the points through which attackers can enter the system, indicating which components can be exploited.

MTD is an extremely broad concept, encompassing virtually all levels of networks and computer systems:

- At the data level, where the data format or representation can be changed;
- At the application level, considering dynamic changes to application code, such as during compilation;
- At the runtime environment level, dynamic changes may include random memory allocation for storing critical data;
- At the operating systems level, utilizing dynamic changes in instruction sets or entire operating systems;
- At the hardware level, involving techniques related to memory, processors, and networks. The first two techniques can utilize methods proposed for operating systems. Techniques used for dynamic changes are much broader and can include, among other things, changes to layer 2 and/or layer 3 addresses, TCP and UDP protocol port numbers, protocol types, and other network parameters.

Changes can be made periodically or randomly. It is also possible to use decoys in the form of servers or stub networks, where the detection of events unrelated to the standard network operations triggers MTD mechanisms. MTD confuses attackers, making it difficult for them to establish a foothold and exploit vulnerabilities in the system.

Implementing MTD mechanisms provides an advantage over attackers by forcing them to confront a constantly evolving and challenging target, thereby increasing the complexity and cost of attacks. This dynamic state is not present in traditional networks, which are characterized by static security measures stemming from fixed configurations and patterns.

5. Protocol-Independent Packet Processors Programming

The first version of the P4 language was developed by a team from Stanford University in 2014 [43]. The first language specification was also published in 2014, and was designated $P4_{14}$. Two years later, in 2016, the $P4_{14}$ standard was replaced by the $P4_{16}$ standard, which is currently in use. The latest version of the $P4_{16}$ standard is version 1.2.4 [44].

The P4 language is a programming language for the data plane in SDNs, meaning it is a programming language for SDN switches (although code written in P4 indirectly affects the control plane as well). Systems that use central processing units (CPUs) are Multi-Instruction stream Multidata stream (MIMD) systems. In contrast, data processing in programs written in the P4 language is of the Multi-Instruction stream Single-Data stream (MISD) type. The differences between these types are presented in Figure 4. Programs and data processing in the P4 language closely resemble the operation of EasyChip network processors, which are divided into Task-Optimized Processors (TOP), specialized for performing specific tasks. To enhance the performance of network processors, individual tasks are parallelized. The TOP processors include the following:

- TOPparser: processors responsible for extracting relevant operations from processed protocol units.
- TOPsearch: processors optimized for data search, such as finding the next hop for a packet or the port number through which a specific MAC address is accessible.
- TOPresolve: processors optimized for making decisions based on data provided by TOPSearch processors.
- TOPmodify: processors optimized for modifying the structure of processed protocol units.

The P4 language specification not only includes the definition of command syntax, control instructions, built-in functions, and data structures but also outlines two types of architectures:

- Portable NIC Architecture (PNA): This architecture describes the common capabilities of Network Interface Cards (NICs) in network devices that process and transmit packets between one or more network interface and the host system.
- PSA: An architecture that details the common capabilities of network switches in terms of packet processing and forwarding.

PNA and PSA architectures facilitate the construction of any computer network (although they do not support the construction of access networks built on specific technologies like Passive Optical Networks (PONs)). They are used to build hosts and network nodes. Each architecture is associated with a defined platform, known as a target. This platform can range from general-purpose CPU, Field Programmable Gate Array (FPGA) devices, specialized Intel Tofino devices, or even Raspberry Pi boards. A significant difference among targets is their achievable performance, which varies from 10 Mbps (for Raspberry Pi devices) to 12 Tbps (when using Intel Tofino devices).

The general PSA architecture, as presented in Figure 5, can be divided into three parts:

- Ingress processing blocks;
- Egress processing blocks;
- A buffering and replication block.

Depending on the target, the number of blocks may be smaller. Not all platforms support the implementation of all processing stages, and implementing all blocks may reduce performance without significantly affecting achievable functionality. The parser blocks shown in the figure are used to extract data relevant to applications. Fields of constant length are straightforward to extract from headers. It is sufficient to define the header of interest by specifying the number of bits the particular field occupies, and individual fields can be retrieved with a single command. However, for variable-length fields, the process is more complex, especially since there are many methods for determining the length of header fields.

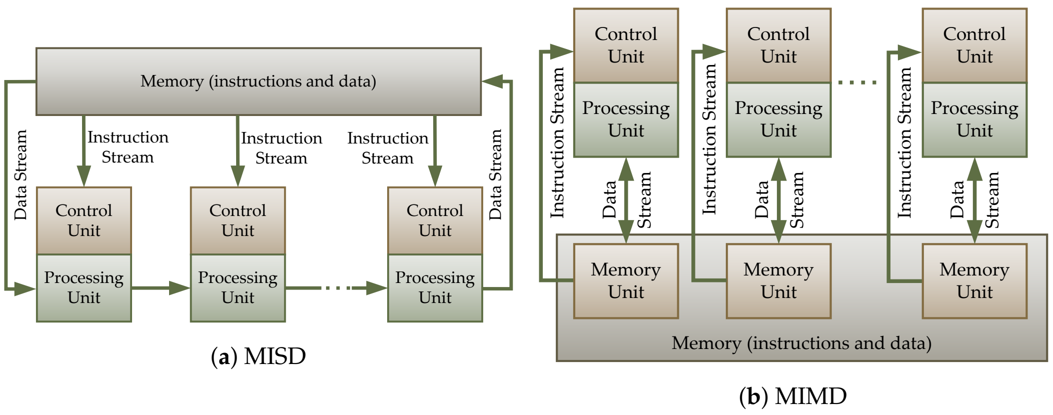


Figure 4. Data and instruction streams in MISD and MIMD systems.

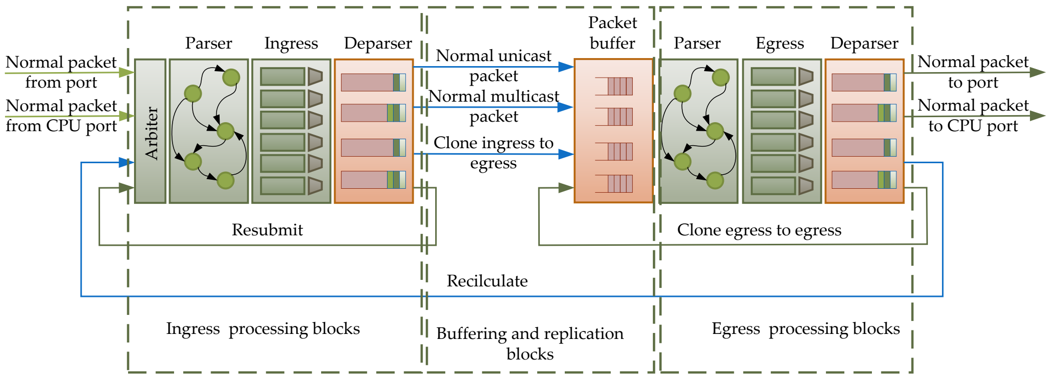


Figure 5. PSA architecture.

The last block in the packet path for both ingress and egress processing is the deparser block. Its task is reassembling the entire message. The data extracted by the parser must be reattached to the portion of the message that has not undergone processing. During deparsing, it is not necessary to add back the exact number of bits that were removed from the packet during the parsing process. This flexibility means that the outgoing packet from the node or NIC card can have not only altered data in the headers but also headers of varying lengths. It is also possible to encapsulate a protocol unit in another protocol.

6. Implementation

6.1. Environment Description

The P4 language specification defines the PSA architecture, which is illustrated in Figure 5. This architecture can be considered a blueprint for P4 switch architecture, and it may vary depending on the device in which it is implemented. Several products on the market facilitate the creation of a data path using the P4 language. Among these are Intel's Application-Specific Integrated Circuit (ASIC) devices, such as Tofino I, Tofino 2, and Tofino 3 [45–47]. Additionally, there is the option to implement an SDN switch in FPGA circuits, for which the special NetFPGA SUME [48], AMD Alveo [49], or Intel [50] platforms have been designed. The models defined for these products vary in the number of blocks on the packet processing path. A significant drawback of all these listed products is their cost. Therefore, in the proposed solution, it was decided to utilize a widely available and free option, the Mininet environment [51], the main component of which is the SimpleSwitch software switch with V1 model implemented. Similar to SimpleSwitch, the V1 model also programmatically realizes the entire pipeline structure [52,53]. The V1 model is depicted in Figure 6. Comparing it to the PSA architecture, shown in Figure 5, differences are easy to identify. The parser block appears only once, at the switch's input, whereas the deparser block is implemented at the switch's output. In contrast, in the PSA architecture, both blocks appear in both the input and output pipelines. Additionally, in the input pipeline, there is also a checksum verification block (IP header), while the output pipeline includes a checksum update block. Both blocks must be utilized when implementing layer 3 functions, where the SDN switch acts as a router. The implementation uses external objects, which can be perceived as built-in functions. Their usage is limited to passing on function call parameters only; therefore, they are not presented in the following description of the implementation.

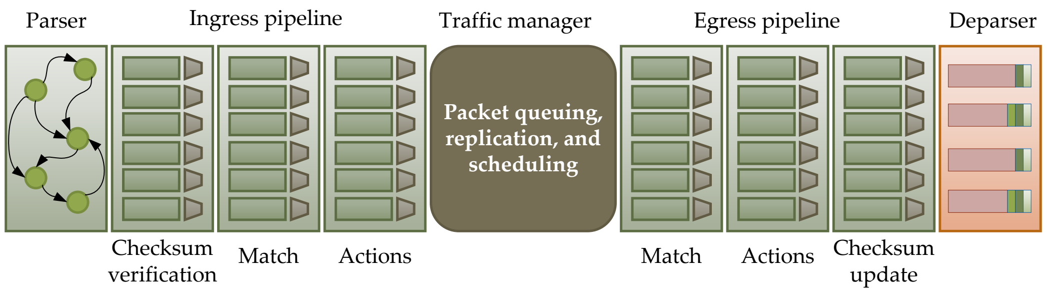


Figure 6. V1 model architecture.

The V1 model architecture is is known as the target architecture for the P4 language. Each target architecture is supported by one or more targets. In the V1 model, the most commonly used target is the Behavioral Model v.2 (BMv2), which is implemented as a programmable switch in the Mininet environment for simulating computer network functions [51,54,55]. It is important to note that not every target architecture may fully implement the P4 language, or the implementation might not fully comply with the language standard. This means that implementing the presented solution may require changes, even at the conceptual level, when used with a different target architecture. Before applying the presented solution to another target architecture, readers are encouraged to familiarize themselves with the BMv2 documentation and the documentation of the selected target architecture.

The BMv2 software switch and Mininet run on PC-class computers and are used for development, testing, and debugging new functionalities, both in the data plane and control plane. Although designed for use in a multithreaded environment, they exhibit significant limitations, affecting its performance. One such limitation is the use of only one thread to handle the input pipeline for all packets arriving at all ports of the switch, while output pipeline processing is, by default, carried out by four threads.

Figure 7 shows the structure of the network, which serves as the basis for preparing algorithms that implement the considered MTD technique. The presented structure includes four hosts and six switches. From the perspective of the functions performed by the switches, we can identify two types of switches:

- Core switches, which are connected only to other switches;
- Border switches, which have connections both to hosts and other switches.

All switches also have a connection to the SDN controller, which controls the operation of the entire network. These connections are only for the transmission of control information, such as entries to flow tables. To simplify the understanding of the algorithm, their participation is not further detailed here. Before describing the functions performed by both types of switches, let us introduce the concept of IP address generation. (Note: In the proposed implementation of the MTD technique, only IPv4 (IP version 4) addresses are used. However, the proposed mechanism can also be applied using IPv6 (IP version 6) addresses. The limitation to IPv4 was chosen to simplify the diagrams and enhance their comprehensibility.)

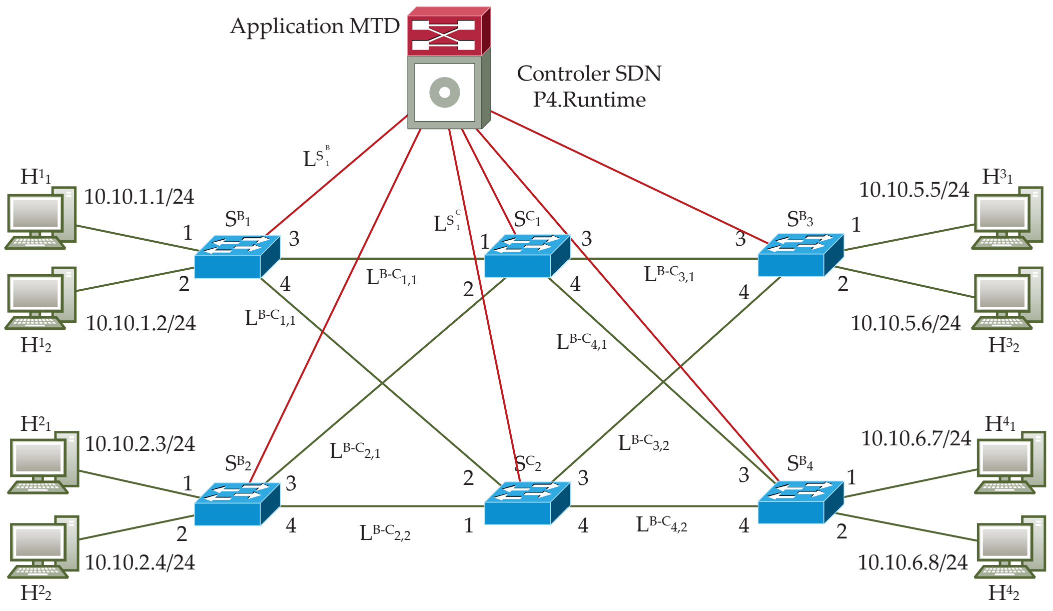


Figure 7. Network structure using lossless MTD mechanisms.

Definition 1. *Define the generation of an IP address as a group of IP addresses that describes an entire computer network secured using MTD mechanisms. These addresses are exchanged in IP packets between border and core nodes, as well as among core nodes, in a specific state of the network.*

Definition 2. *Let us define a coherent network state as a state in which all border devices use only one logically consistent set of IP addresses.*

The dynamic mutation of IP addresses in computer networks can proceed in two ways, synchronously and asynchronously. In synchronous mode, all network nodes should change the used group of IP addresses simultaneously. This necessitates the introduction of network synchronization mechanisms. An example of synchronous network parameter change can be observed in passive optical networks, where the Next-Generation Passive Optical Network 2 (NG-PON2) standard includes a mechanism for dynamically changing the wavelength used for data transmission between Optical Line Termination (OLT) and Optical Network Unit (ONU). To indicate the moment when the wavelength used by the ONU should change, a superframe counter is utilized, which is increased with each transmitted frame. Since only two devices (the OLT and one ONU) participate in synchronization, this mechanism works reliably. The structure of computer networks is much more complex than that of NG-PON2 networks, which only have point-to-multipoint connections. In computer networks, many nodes connect to many nodes, so achieving synchronization using the method used in NG-PON2 is virtually impossible. Another method of achieving synchronization is the use of synchronous ethernet, described in [56]. Synchronous ethernet does not address the issue of changing IP addresses, which would not affect established TCP connections. Synchronous ethernet only offers synchronization at the bit or byte level. A problem arises if IP address changes occur during the transmission of an ethernet frame: a node might begin transmitting an ethernet frame containing an IP packet with a specific IP address, and, if IP addresses in the network change during transmission, the IP addresses upon reception could fall outside the correct address space. While it might be feasible to introduce a protective period to complete the transmission of the started frames, determining the precise duration of this protective period would be challenging. This duration would need to account for frame propagation times, their processing in nodes, etc., for any pair of nodes. It is possible to assume a sufficiently long protective time, but this would likely result in a deterioration in service quality (reduced network throughput, extended transmission time). Furthermore, periodic breaks in data transmission would be a signal to an observer that some change in the network structure is taking place. To avoid problems with synchronous mode, the proposed solution uses an asynchronous method. In this approach, network nodes are allowed to use two generations of IP addresses—old and new—for a certain period. To make this possible, the following requirements must be met: Before using the new generation of IP addresses, it is necessary to distribute all necessary entries related to the new generation in the flow tables in all network nodes, i.e., in both border and core nodes. The start of using the new generation of IP addresses occurs when the SDN controller introduces information about the new generation into the appropriate flow table in at least one network node. In this scenario, the process of changing generations throughout the entire network lasts for a specified period. Unlike the wavelength-changing methods in NG-PON2 networks, it is impossible to precisely predict when information about the next generation of IP addresses will reach the switch. The change in the current generation in the switch can occur in two way: either by an entry into the flow table or by the switch detecting the new generation by the switch during packet processing. This necessitates storing information about the generation not only in the flow table but also in a structure the data plane's program code can modify. This structure could be a register. Each target of the P4 language offers several registers that can be utilized in the program code. One feature of registers is their ability to store data between subsequent packet processing events, meaning that changes introduced in the register during the processing of packet n are available during the processing of packet $n + 1$. Each switch, when detecting a new generation of IP addresses, modifies the contents of the register. If the contents of the register contain a newer (different) generation, it sends a message to the SDN controller. The removal of entries related to the outdated generation of IP addresses from flow tables is possible when the network reaches a specific state, i.e., when each switch in the network receives an entry from the SDN controller to the appropriate flow table about the new version or detects the new generation during packet processing and successfully informs the SDN controller about it. Immediately after removing outdated (old) entries from the flow tables, the network should be prepared for the use of the next generation of IP addresses by delivering new entries to all switches. This preparation involves an algorithm for controlling the core switch. Core switches aggregate traffic from many border switches. Their functions are limited to determining to which port of the switch the packet should be sent.

6.2. Lossless Dynamic IP Address Mutation Algorithms

As mentioned in the previous subsection, the network comprises two types of network elements: core switches and border switches. In the proposed solution, both types of switches implement a different algorithm. For core switches, the algorithm (illustrated in Figure 8 and, as pseudocode, in Algorithm 1) is confined to directing the packet to the appropriate output interface or deleting the packet in the absence of a corresponding entry. Since the entire network is assumed to use IP addresses assigned by the SDN controller, the appearance of a packet directed to an IP address not within the group of addresses assigned to hosts can, at most, activate functions that inform the controller about such an event. It is crucial that before adopting a new generation of IP addresses, the flow tables of all core switches contain all the necessary entries related to that generation. Core switches do not modify packet headers, except for the standard operation of the IP protocol, namely, the decrease in the Time-To-Live (TTL) field. Since this is a standard operation, it is not included in Figure 8 or Algorithm 1. The processing algorithm us presented in two ways, as a diagram and as pseudocode, enabling readers to more easily grasp the operation of the proposed method. While the diagram does not indicate all the keys used in searching the flow tables, it significantly aids in analyzing the algorithm.

Algorithm 1: Packet processing algorithm in core switch.

```
Data: headers struct, standardmetadata struct
Result: output packet
$match\_key \leftarrow (headers.IPv4.dest\_addr)$;
if $coreIPv4Table.apply(match\_key) == acction\_ForwardPacket()$ then
    Set output port based on parameters from $acction\_ForwardPacket()$;
else
    Mark packet to drop;
end
```

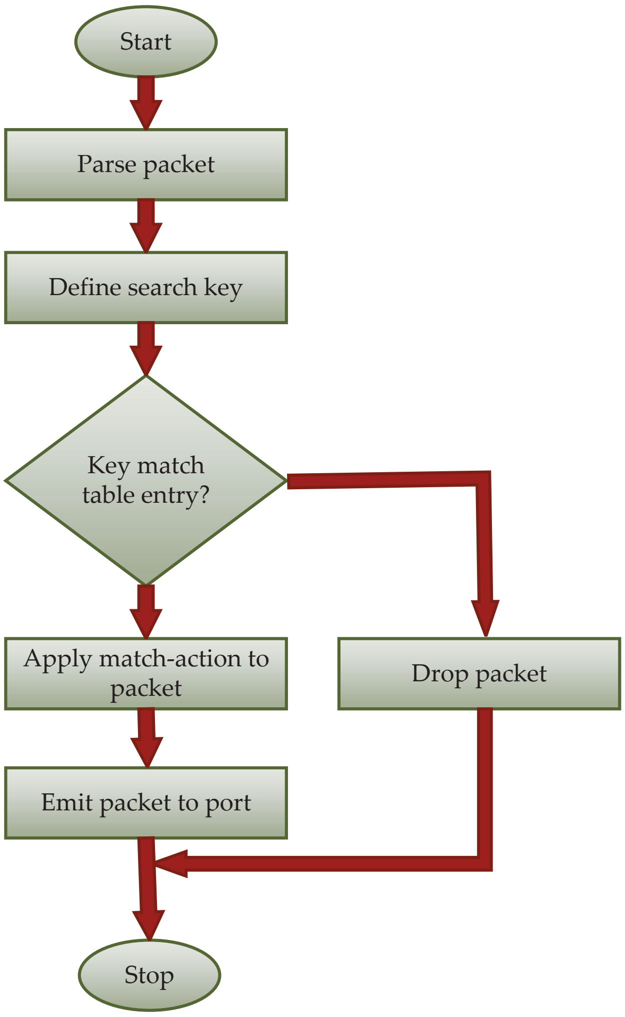


Figure 8. Packet processing algorithm in a core switch.

In the case of border switches, whose algorithms are illustrated in Figure 9 and Algorithm 2, the processing process includes a larger number of conditional instructions. The common step in all paths of the algorithm is determining the input interface and the interface through which the packet must be sent. The number of the input interface is already provided in the parser block as standard metadata. The number of the output interface can only be determined in the ingress match block after searching the appropriate flow table. Input and output interfaces can be categorized into local and core interfaces. Based on the pair <input interface, output interface>, we can identify four packet processing processes:

- <local, local>: In this mode, the packet is directed to a specific local interface. The only modification that may occur is updating the TTL field and the checksum field of the IP header.

- <local, core>: In this scenario, it is necessary to modify both the source and destination IP addresses, which are contained in the respective flow tables. The search involves IP addresses and a value representing the current generation, which can be stored in a separate flow table or register (register is a special type of data in the P4 language, and a variable, uniquely retains a value determined during the processing of the previous packet).
- <core, local>: The scenario very similar to the <local, core> case. The modification of the IP addresses to those used in the local network is also required. To find the IP addresses that will replace the IP addresses present in the packet headers, only the IP addresses from the header are utilized. The result of the search also returns the generation of IP addresses. If the searched generation is newer than the current generation, it indicates that the system has started using a new generation. In such a case, modifying the generation table or register and sending a message to the SDN controller once is necessary.
- <core, core>: This scenario, not included in the diagrams to avoid further expanding them, is possible to implement. The operation mirrors that of the <local, local> case, albeit with different flow tables.

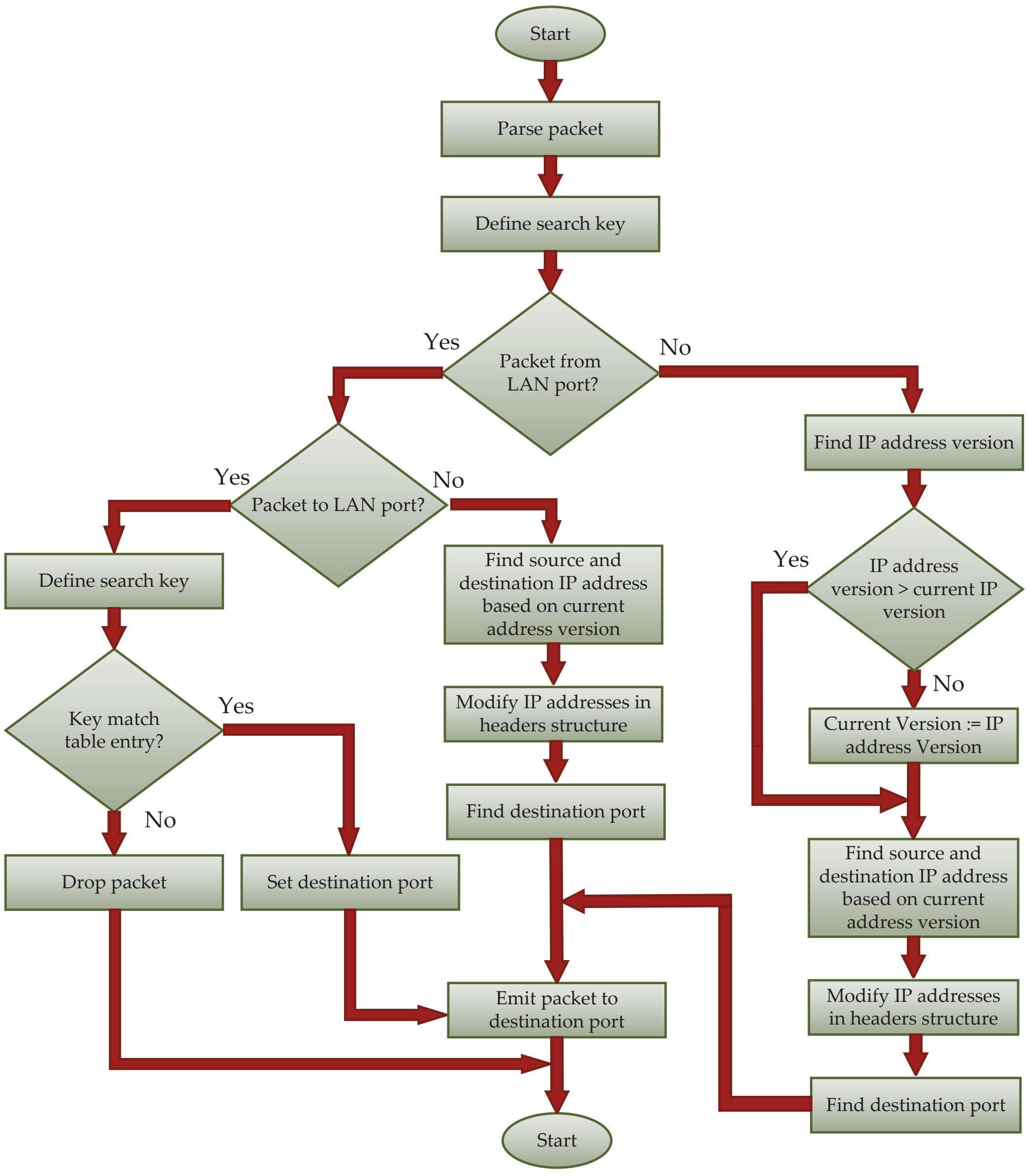

Start
Parse packet
Define search key
Packet from LAN port?
Yes
No
Find IP address version
Packet to LAN port?
Yes
No
Define search key
Find source and destination IP address based on current address version
IP address version > current IP version
Yes
No
Current Version := IP address Version
Key match table entry?
Yes
No
Modify IP addresses in headers structure
Find destination port
Drop packet
Set destination port
Find source and destination IP address based on current address version
Modify IP addresses in headers structure
Emit packet to destination port
Find destination port
Start

Figure 9. Packet processing algorithm in an border switch.

Algorithm 2: Packet processing algorithm in border switch.

```
Data: headers struct, standardmetadata struct
Result: output packet
$match\_key \leftarrow (headers.IPv4.src\_addr)$;
if $borderLanPortSrcTable.apply(match\_key) == acction\_PacketFromLanPort()$
 then
    $match\_key \leftarrow (headers.IPv4.dest\_address)$;
    if $borderLanDstTable.apply(match\_key) == acction\_PacketToLanPort()$ then
        $match\_key \leftarrow (headers.IPv4.dst\_addr)$;
        if $borderLanPortDstTable.apply(match\_key)$ then
            Set destination port based on action parameters;
        else
            Mark packet to drop;
        end
    else
        $match\_keyA \leftarrow (headers.IPv4.dst\_addr, current_IP_version)$;
        $match\_keyB \leftarrow (headers.IPv4.src\_addr, current_IP_version)$;
        Translate destination IP address and set output port number according to
          $acction\_TranslateDstAddress()$ executed by
          $borderTranslationDstTable.apply(match\_keyA)$;
        Translate source IP address according to $acction\_TranslateSrcAddress()$
          executed by $borderTranslationSrcTable.apply(match\_keyB)$
    end
else
    $match\_key \leftarrow (headers.IPv4.src\_addr)$;
    Get $IP\_address\_version$ from $acction\_PacketFromLanPort()$ executed by
      $borderLanPortSrcTable.apply(match\_key)$;
    if $IP\_address\_version > current\_IP\_version$ then
        $current\_IP\_version \leftarrow IP\_address\_version$;
    end
    $match\_keyA \leftarrow (headers.IPv4.dst\_addr, current\_IP\_version)$;
    $match\_keyB \leftarrow (headers.IPv4.src\_addr, current\_IP\_version)$;
    Translate destination IP address and set output port number according to
      $acction\_TranslateDstToLanAddress()$ executed by
      $borderTranslationDstToLanTable.apply(match\_keyA)$;
    Translate source IP address according to $acction\_TranslateSrcToLanAddress()$
      executed by $borderTranslationSrcToLanTable.apply(match\_keyB)$
end
if Packet is not marked to drop then
    Emit Packet
else
    Drop Packet
end
```

Implementing the proposed mechanism in traditional computer networks is practically impossible. Routers make packet routing decisions based on the entries in the routing table, the contents of which are supplied by specific routing protocols. Changing the contents of routing tables is possible, for example, by using static entries that would be supplied through the Representational State Transfer Configuration (RESTCONF) protocol [57]. However, such a procedure would be quite time consuming. This means that services provided by networks would be temporarily unavailable for a certain period, which, in modern networks with throughput expressed in tens of GB/s, is unacceptable. Moreover, changing the IP addresses of router interfaces forces the change in addresses for all hosts connected to the router (directly or via layer 2 switches). This requirement could be addressed by using the DHCP protocol. However, introducing another traditional mechanism would further extend the service unavailability. Even using NAT functions would not solve the problem due to functional limitations (changing IP addresses affects existing TCP sessions, problems with UDP protocol handling, the need for port forwarding). Implementing such a solution in a traditional computer network, from the network administrator's perspective, would be an unimaginable endeavor.

7. Performance Evaluation

7.1. Convergence Time

To conduct the optimization process and assess the effectiveness of the solution, it is necessary to determine the system's convergence time, meaning the time required for all switches to exclusively use addresses from the current generation (this also includes the IP addresses of packets in the network interface buffers and packet memories). Two scenarios should be considered.

Scenario I: All modification are introduced simultaneously by C^{SDN} in all border switches. In this scenario, the convergence time is determined by two components:

- The time to update the flow tables in all border switches ($T^{Controler}$);
- The longest packet transfer time between two border switches ($T^{Transfer}$).

For formal notation, let us introduce designations for the active elements (such as core and border switches, hosts, and the SDN controller) and passive elements, namely, links. Let S_i^C denote the ith core switch. Similarly, let us denote the jth border switch as S_j^B. Host number k, connected to S_j^B, is denoted as H_k^j. The SDN controller, having only one instance, is simply denoted as C^{SDN}. The link between $S_{i_1}^C$ and $S_{i_2}^C$ (i.e., the link between the i_1th and i_2th core switches) is denoted as L_{i_1,i_2}^{C-C}, the link between $S_{j_1}^B$ and $S_{j_2}^B$ (i.e., the link between the j_1th and j_2th border switches) is denoted as L_{j_1,j_2}^{B-B}, while the link between H_k^j and S_j^B is denoted as $L_{i,j}^{C-B}$. The link between the controller and $S_{i_1}^C$ or $S_{i_2}^B$ is denoted as $L^{S^C_{i_1}}$ or $L^{S^B_{i_2}}$, respectively. Figure 7 illustrates these network elements along with their designations.

Additionally, let us introduce the notation of the function $T(x)$, which returns the time value in which an IP packet is in element (which can be either an active and a passive element) x.

The first component of convergence time is defined by the following equation:

$$T^{Controler} = \max\left(T\left(I^{C_i^{\mathrm{SDN}}}\right) + T\left(I^{C_j^{\mathrm{SDN}}}\right) + T\left(L^{S_i^C}\right) + T\left(I^{S_j^B}\right)\right), \tag{1}$$

where $T\left(I^{C_i^{\mathrm{SDN}}}\right)$ ($T\left(I^{C_j^{\mathrm{SDN}}}\right)$) is the time of generation, buffering, and sending a message in the SDN controller on the interface to S_i^C (S_j^B); $T\left(I^{S_j^B}\right)$ is the time of receiving, verifying correctness, buffering, and updating the flow table in switch S_j^B; and $T\left(L^{S^C i_1}\right)$ is the propagation time between C^{SDN} and S_j^B. Assuming that C^{SDN} sends almost identical messages to all S_j^B (only differing in IP and MAC addresses), that all interfaces of C^{SDN} are identical, and that all S_j^B switches are identical, Equation (1) can be simplified to

$$T^{Controler} = T\left(I^{C^{\mathrm{SDN}}j}\right) + T\left(I^{S_j^B}\right) + \max\left(T\left(L^{S^B j}\right)\right). \tag{2}$$

This means that the time of the first component is equal to the time of updating the flow table in the border switch most distant from C^{SDN}.

The second component of the convergence time is equal to the longest duration for an IP packet to pass between two edge switches. Excluding situations where a packet may be buffered in switches (due to temporary disconnection/switching of the network link), unusually long packet handling in the switch, etc., this time can be calculated from

$$T^{Transfer} = \max\left(T\left(I^{S^B_{j_1}}\right) + T\left(I^{S^B_{j_2}}\right) + T\left(L^{C-B}_{i_1,j_1}\right) + T\left(L^{C-B}_{i_2,j_2}\right) + \right. \\ \left. + \sum T\left(S^C_i\right) + \sum T\left(L^{C-C}_{i_1,i_2}\right)\right) \tag{3}$$

where $T\left(I^{S^B_{j_1}}\right)$ is the time of generating and sending an IP packet from $S^B_{j_1}$, while $T\left(I^{S^B_{j_2}}\right)$ is the time of receiving and processing an IP packet in $S^B_{j_1}$. The times $T\left(L^{C-B}_{i_1,j_1}\right)$ and $T\left(L^{C-B}_{i_2,j_2}\right)$ are the propagation times on the links connecting border switches to core switches. The time $T\left(S^C_i\right)$ is the time of receiving the packet, processing it (finding the route for the packet, modifying IP header fields), and sending it to the next switch. The time $T\left(L^{C-C}_{i_1,i_2}\right)$ is the propagation time between two core switches. Similar to $T^{Controler}$, we can assume that all active elements are identical, so Formula (3) can be simplified to

$$T^{Transfer} = T\left(I^{S^B j_1}\right) + T\left(I^{S^B_{j_2}}\right) + \min\left(T\left(L^{C-B}_{i_1,j_1}\right) + T\left(L^{C-B} i_2, j_2\right) + \right. \\ \left. + c \cdot T\left(S^C_i\right) + \sum T\left(L^{C-C}_{i_1,i_2}\right)\right) \tag{4}$$

where c is the number of core switches located on the path between two border switches. The signal in a fiber optic link is transmitted at the speed of light ($3 \cdot 10^8$ m/s), which means that sending a frame over a distance of 1 km takes 3.3 µs. Assuming that the protected network is located in a limited area, where the length of links between core switches is similar and does not exceed the length DL (e.g., 1 km), Formula (4) can be simplified to

$$T^{Transfer} = T\left(I^{S^B_{j_1}}\right) + T\left(I^{S^B_{j_2}}\right) + \min\left(T\left(L^{C-B}_{i_1,j_1}\right) + T\left(L^{C-B}_{i_2,j_2}\right) + \right. \\ \left. + c \cdot T\left(S^C_i\right) + (c-1) \cdot T(DL)\right) \tag{5}$$

Considering all time components, the convergence time in scenario I is calculated as

$$T^I_{\text{convergence}} = T^{Controler} + T^{Transfer} \tag{6}$$

Scenario II: In this scenario, it is assumed that the SDN controller cannot send information about the change in IP address generation to all switches simultaneously. Instead, the data exchange operation follows a sequential algorithm, where the controller only proceeds to modify the flow tables in the next switch after receiving confirmation of the modification's completion from the currently modified switch. The controller can proceed to modify the flow tables in the next switch only after receiving confirmation of the completion of the modification from the switch currently being modified. According to Algorithm 2, the change in IP address generation, enforced by messages from the SDN controller, can be supplemented by propagating information about the new generation through the exchange of IP packets in the data plane. It is worth noting a potential Scenario III, where the SDN controller informs only one switch about the generation change, and the other switches adjust the generation of IP addresses based on packet exchanges in the data plane. This approach has a significant limitation: if a given edge switch does not receive IP packets with addresses from the new generation—or receives no IP packets at all—it will continue using the previous generation of IP addresses. While using generation propagation in the data plane can speed up the generation change in individual switches, informing all switches through the SDN controller is necessary to achieve a converged state.

In such a case, the convergence time in scenario II (denoted as $T^{II}_{\text{convergence}}$) is equal to the sum of the message transfer times between the SDN controller and all SDN switches, as well as the longest packet transmission time between two border switches. $T^{II}_{\text{convergence}}$ is given by

$$T^{II}_{\text{convergence}} = (c + b) \cdot T^{Controller} + T^{Transfer} \tag{7}$$

where c is the number of core switches, and b is the number of border switches.

7.2. Numerical Experiment

In Section 7.1, considerations regarding the convergence time of the system were presented, which is the time required for all switches to use only the set of IP addresses belonging to a given generation. Table 1 presents the time characteristics of the SDN elements that are necessary to calculate the convergence times in scenarios I and II. Figure 10 compares the convergence times of networks using scenarios I and II for different link bandwidths (10 Gbps and 40 Gbps) and different numbers of switches. The chart shows the total number of border and core switches. It was assumed that the ratio of core to border switches was 1:5. As can be observed, the convergence time did not exceed 350 µs.

Table 1. Time characteristics of SDN elements.

System Element	Processing Time
Serialization on ethernet port (1 Gbps, packet size 1518 bytes) [58]	7.2 µs
Serialization on ethernet port (10 Gbps, packet size 1518 bytes) [58]	1.2 µs
Serialization on ethernet port (25 Gbps, packet size 1518 bytes) [58]	0.48 µs
Serialization on ethernet port (40 Gbps, packet size 1518 bytes) [58]	0.3 µs
IP packet transmission over a distance of 1 km [59]	∼5 µs
Packet forwarding in SDN switch (depends on ASIC) [60]	∼0.2 µs
Flow table update (500 Mbps, packet size 1500B) [61]	∼90 ms

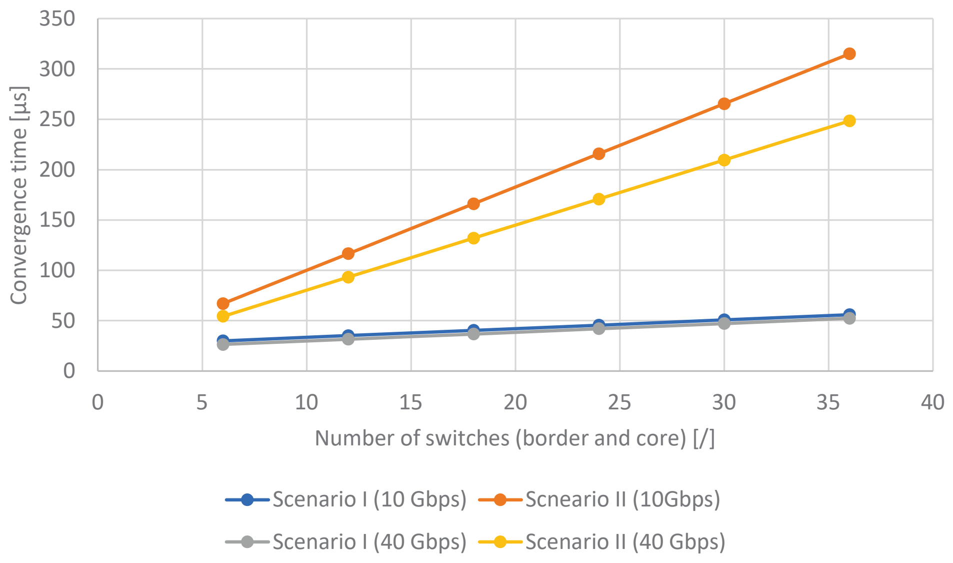


Figure 10. Comparison of convergence times in scenarios I and II for different link bandwidths and different numbers of switches.

In order to assess the performance of the proposed solution, we needed to consider the time required to prepare the switches for the application of IP addresses of the next generation. For all switches to be able to use the new generation, the SDN switch must update the flow tables in all switches. According to Algorithms 1 and 2, it is necessary to modify the tables:

- $coreIPV4Table$ in core switches, which contains h^2 entries, where h is the number of hosts in the supported network,
- $borderTanslationDstTabl$ in border switches, which contains h entries;

- *borderTanslationSrcTabl* in border switches, which also contains h entries.

The time required to modify flow tables is dependent on the amount of modifying data, which in turn depends on the number of commands and parameters contained in the packet [61]. Modifying commands are transmitted between the SDN controller and the switches as text, formatted according to JSON notation. The number of bytes in such a command depends on the number of characters. However, it should be noted that determining the exact number is impossible, as, for example, an IP address in version 4 is made up of 4 bytes, but in JSON notation, it can occupy from 7 bytes (e.g., 1.1.1.1) to 15 bytes (e.g., 192.168.136.200). The names of tables and the functions called also affect the number of characters transmitted in a single command. For simplicity, let us assume that one command is 60 bytes (characters) long.

Let us determine the time necessary to update a table involving the exchange of R rows. Based on the research results presented in [61], we can assume that updating 25 entries (according to the assumption, the modifying command is 60 bytes long) in a table takes approximately 90 ms. Let $T^{modify}(R)$ denote the time needed to modify R rows in the flow table. $T^{modify}(R)$ can be determined using

$$T^{modify}(R) = R \cdot T^{mod}, \tag{8}$$

Taking into account the modification times presented in [61], Formula (8) can be simplified to the form

$$T^{modify}(R) = 90\left\lceil \frac{R}{25} \right\rceil, \tag{9}$$

Meanwhile, the time required to modify tables (denoted as $T^{prepare}(k, b, c)$) in all the switches of the considered network, which is composed of h hosts connected to each border switch, b border switches, and c core switches, is

$$T^{prepare}(k, b, c) = b \cdot T^{modify}(k) + b \cdot T^{modify}(k(b-1)) + c \cdot T^{modify}((b \cdot k)^2) \tag{10}$$

In Figure 11, the changes in the time necessary for updating the flow tables in all network switches are presented. It was assumed that each border switch was connected to 24 hosts. Meanwhile, the ratio of the number of core hosts to the number of border switches was 1:5 (similar to the comparison presented in Figure 10). As can be observed, the time required to prepare the switches for the change in IP address generation increases exponentially with the increase in the number of switches (and consequently with the increase in the number of hosts) from a few seconds to 3 h. It should be noted that this solution was not optimized in this respect.

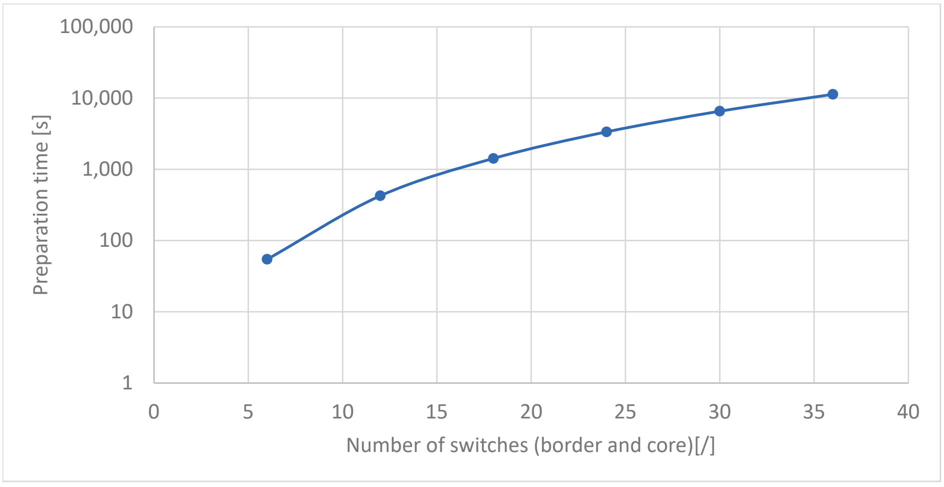


Figure 11. Comparison of switch preparation time for different number of border and core switches.

8. Conclusions and Future Works

Our civilization's dependence on IT solutions, as well as the geopolitical situation, requires new research tasks and the delivery of new solutions in the field of cybersecurity. This article proposed a new approach to one of the MTD technologies using IP address mutation. We used an SDN and programming in the P4 language, which adds a new dimension to shaping the functionality of the data plane. In research on the effectiveness of MTD mechanisms, studies have rarely been conducted on the impact of the applied techniques on the quality of services provided to users [62]. The use of MTD techniques can lead to disruptions in service availability [7,8] or, as is the case with IP address mutation mechanisms that utilize DHCP or DNS services, to connection interruptions [9]. The presented solution is distinguished from other proposed solutions by its ability to maintain the continuity of services. By applying the described solution, any interruption in TCP connections is avoided, which is significant for networks operating at bandwidths above 10 Gbps. The inability to transmit data even for a few seconds can result in significant data loss. Another important advantage of the proposed solution is its flexibility. This solution does not require modifications to the operation of services such as DHCP or DNS and thus is not limited by their functioning.

Performance analysis of the proposed solution indicated that with an increasing number of hosts in the network, the permissible speed of IP address mutation drastically decreases. This is due to the use of an extremely simple mutation algorithm, in which IP addresses were mutated in a completely random manner at each step, necessitating the modification of entire flow tables. Such an algorithm is not optimal. In further work, researchers can attempt to optimize the mutation algorithm, in which IP addresses will be changed only for a portion of the hosts in subsequent mutations. Another scenario to consider is mutating addresses only for devices providing critical services. Considering the flexibility of implementation provided by the P4 language, further studies can introduce additional mutation scenarios.

Another direction for future research involves conducting a detailed analysis of the proposed solution. The Mininet environment used in this study only allowed for the confirmation of the proposed solution's functionality. Since preparing a sufficiently large research environment, where programmable SDN switches would be utilized, is an extremely difficult and costly endeavor, there is a plan to develop an appropriate simulator in the OMNET++ environment.

Author Contributions: M.Ż., M.M. and P.Z.: conceptualization, validation, writing—review and editing; M.Ż.: data curation, formal analysis, investigation, methodology, resources, software, visualization, and writing—original draft; P.Z.: funding acquisition, project administration, supervision. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Polish Ministry of Science and Higher Education (No. 0313/SBAD/1310).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

API	Application Programming Interface
APT	Advanced Persistent Threat
ASIC	Application-Specific Integrated Circuit
BGP	Border Gateway Protocol
BMv2	Behavioral Model v.2

CKC	Cyber Kill Chain
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
FPGA	Field Programmable Gate Array
IGRP	Interior Gateway Routing Protocol
IP	Internet Protocol
IS-IS	Intermediate System to Intermediate System
IT	Information Technology
JSON	JavaScript Object Notation
MAC	Media Access Control
MIMD	Multi-Instruction stream Multidata stream
MISD	Multi-Instruction stream Single-Data stream
MTD	Moving Target Defense
NAT	Network Address Translation
NG-PON2	Next-Generation Passive Optical Network 2
NIC	Network Interface Card
ONF	Open Networking Foundation
OSPF	Open Shortest Path First
ONU	Optical Network Unit
OLT	Optical Line Termination
OSINT	Open Source Intelligence
P4	Protocol-Independent Packet Processor Programming
PON	Passive Optical Network
PNA	Portable NIC Architecture
PSA	Portable Switch Architecture
QoS	Quality of Service
RESTCONF	Representational State Transfer Configuration
SDN	Software Defined Network
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
TOP	Task-Optimized Processors
TTL	Time-To-Live
UDP	User Datagram Protocol

References

1. Safjański, T.; Łabez, P. Counter-Detection Activities Of Criminal Organizations Aimed At Reducing The Effectiveness Of Surveillance Conducted As Part Of Law Enforcement Operational Activities. *Issues Forensic Sci.* **2017**, *298*, 62–68. [CrossRef]
2. Alani, M.M.; Damiani, E. XRecon: An Explainbale IoT Reconnaissance Attack Detection System Based on Ensemble Learning. *Sensors* **2023**, *23*, 5298. [CrossRef] [PubMed]
3. Grigaliūnas, v.u.; Brūzgienė, R.; Venčkauskas, A. The Method for Identifying the Scope of Cyberattack Stages in Relation to Their Impact on Cyber-Sustainability Control over a System. *Electronics* **2023**, *12*, 591. [CrossRef]
4. Belalis, I.; Spathoulas, G.; Anagnostopoulos, I. Modeling Intruder Reconnaissance Behavior through State Diagrams to Support Defensive Deception. *J. Cybersecur. Priv.* **2023**, *3*, 275–302. [CrossRef]
5. Huang, Y.T.; Lin, C.Y.; Guo, Y.R.; Lo, K.C.; Sun, Y.S.; Chen, M.C. Open Source Intelligence for Malicious Behavior Discovery and Interpretation. *IEEE Trans. Dependable Secur. Comput.* **2022**, *19*, 776–789. [CrossRef]
6. Jalowski, L.; Zmuda, M.; Rawski, M. A Survey on Moving Target Defense for Networks: A Practical View. *Electronics* **2022**, *11*, 2886. [CrossRef]
7. Han, Y.; Lu, W.; Xu, S. Characterizing the Power of Moving Target Defense via Cyber Epidemic Dynamics. *arXiv* **2014**, arXiv:1404.6785.
8. Wright, M.; Venkatesan, S.; Albanese, M.; Wellman, M. Moving Target Defense against DDoS Attacks: An Empirical Game-Theoretic Analysis. In Proceedings of the CCS'16: 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24 October 2016; Volume 10, pp. 93–104. [CrossRef]
9. Clark, A.; Sun, K.; Poovendran, R. Effectiveness of IP address randomization in decoy-based moving target defense. In Proceedings of the 52nd IEEE Conference on Decision and Control, Firenze, Italy, 10–13 December 2013; pp. 678–685. [CrossRef]
10. Javadpour, A.; Ja'fari, F.; Taleb, T.; Shojafar, M.; Yang, B. SCEMA: An SDN-Oriented Cost-Effective Edge-Based MTD Approach. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 667–682. [CrossRef]

11. Yan, J.; Zhou, Y.; Wang, T. A Port-Hopping Technology against Remote Attacks and Its Effectiveness Evaluation. *Electronics* **2023**, *12*, 2477. [CrossRef]
12. Chang, S.Y.; Park, Y.; Ashok Babu, B.B. Fast IP Hopping Randomization to Secure Hop-by-Hop Access in SDN. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 308–320. [CrossRef]
13. Volckaert, S. Randomization-based Defenses against Data-Oriented Attacks. In Proceedings of the 8th ACM Workshop on Moving Target Defense, MTD'21, Virtual Event, Republic of Korea, 15 November 2021; pp. 1–2. [CrossRef]
14. Thompson, M.; Mendolla, M.; Muggler, M.; Ike, M. Dynamic Application Rotation Environment for Moving Target Defense. In Proceedings of the 2016 Resilience Week (RWS), Chicago, IL, USA, 16–18 August 2016; pp. 17–26. [CrossRef]
15. Morphisec. *Optimizing the Security Stack with Morphisec and Windows Defender*; Whitepaper; Morphisec Labs: Beersheba, Israel, 2020. Available online: https://www.morphisec.com/hubfs/Optimizing-Sec-Stack-Morphisec-Dfndr-200929.pdf (accessed on 30 October 2023).
16. Okhravi, H.; Comella, A.; Robinson, E.; Haines, J. Creating a cyber moving target for critical infrastructure applications using platform diversity. *Int. J. Crit. Infrastruct. Prot.* **2012**, *5*, 30–39. : 10.1016/j.ijcip.2012.01.002 [CrossRef]
17. Salamat, B.; Gal, A.; Jackson, T.; Manivannan, K.; Wagner, G.; Franz, M. Multi-variant Program Execution: Using Multi-core Systems to Defuse Buffer-Overflow Vulnerabilities. In Proceedings of the 2008 International Conference on Complex, Intelligent and Software Intensive Systems, Barcelona, Spain, 4–7 March 2008; pp. 843–848. [CrossRef]
18. Okhravi, H.; Riordan, J.; Carter, K. Quantitative Evaluation of Dynamic Platform Techniques as a Defensive Mechanism. In Proceedings of the Research in Attacks, Intrusions and Defenses, Gothenburg, Sweden, 17–19 September 2014; pp. 405–425.
19. Rawski, M. Network Topology Mutation as Moving Target Defense for Corporate Networks. *Int. J. Electron. Telecommun.* **2019**, *65*, 571–577. [CrossRef]
20. Wang, L.; Wu, D. Moving Target Defense Against Network Reconnaissance with Software Defined Networking. In Proceedings of the Information Security, Honolulu, HI, USA, 3–6 September 2016; pp. 203–217.
21. Hong, J.B.; Yoon, S.; Lim, H.; Kim, D.S. Optimal Network Reconfiguration for Software Defined Networks Using Shuffle-Based Online MTD. In Proceedings of the 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS), Hong Kong, China, 26–29 September 2017; pp. 234–243. [CrossRef]
22. Steinberger, J.; Kuhnert, B.; Dietz, C.; Ball, L.; Sperotto, A.; Baier, H.; Pras, A.; Dreo, G. DDoS defense using MTD and SDN. In Proceedings of the NOMS 2018—2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 23–27 April 2018; pp. 1–9. [CrossRef]
23. Luo, Y.B.; Wang, B.S.; Wang, X.F.; Zhang, B.F. A keyed-hashing based self-synchronizationmechanism for port address hopping communication. *Front. Inf. Technol. Electron. Eng.* **2017**, *18*, 719. [CrossRef]
24. Zhang, L.; Wei, Q.; Gu, K.; Yuwen, H. Path hopping based SDN network defense technology. In Proceedings of the 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Changsha, China, 13–15 August 2016; pp. 2058–2063. [CrossRef]
25. Aydeger, A.; Saputro, N.; Akkaya, K.; Rahman, M. Mitigating Crossfire Attacks Using SDN-Based Moving Target Defense. In Proceedings of the 2016 IEEE 41st Conference on Local Computer Networks (LCN), Dubai, United Arab Emirates, 7–10 November 2016; pp. 627–630. [CrossRef]
26. Zhao, Z.; Gong, D.; Lu, B.; Liu, F.; Zhang, C. SDN-Based Double Hopping Communication against Sniffer Attack. *Math. Probl. Eng.* **2016**, *2016*, 8927169. [CrossRef]
27. Hyder, M.F.; Fatima, T. Towards Crossfire Distributed Denial of Service Attack Protection Using Intent-Based Moving Target Defense Over Software-Defined Networking. *IEEE Access* **2021**, *9*, 112792–112804. [CrossRef]
28. Zhou, Z.; Xu, C.; Kuang, X.; Zhang, T.; Sun, L. An Efficient and Agile Spatio-Temporal Route Mutation Moving Target Defense Mechanism. In Proceedings of the ICC 2019—2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6. [CrossRef]
29. Chowdhary, A.; Alshamrani, A.; Huang, D.; Liang, H. MTD Analysis and evaluation framework in Software Defined Network (MASON). In *SDN-NFV Sec'18: Proceedings of the Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*; ACM: New York, NY, USA, 2018; pp. 43–48. [CrossRef]
30. Wang, J.; Xiao, F.; Huang, J.; Zha, D.; Hu, H.; Zhan, H. CHAOS: An SDN-based Moving Target Defense System. *Secur. Commun. Netw.* **2017**, *2017*, 3659167. [CrossRef]
31. Luo, Y.B.; Wang, B.S.; Wang, X.F.; Hu, X.F.; Cai, G.L.; Sun, H. RPAH: Random Port and Address Hopping for Thwarting Internal and External Adversaries. In Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki, Finland, 20–22 August 2015; Volume 1, pp. 263–270. [CrossRef]
32. Macwan, S.; Lung, C.H. Investigation of Moving Target Defense Technique to Prevent Poisoning Attacks in SDN. In Proceedings of the 2019 IEEE World Congress on Services (SERVICES), Milan, Italy, 8–13 July 2019; Volume 2642-939X, pp. 178–183. [CrossRef]
33. Sharma, D.P.; Kim, D.S.; Yoon, S.; Lim, H.; Cho, J.H.; Moore, T.J. FRVM: Flexible Random Virtual IP Multiplexing in Software-Defined Networks. In Proceedings of the 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing and Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), New York, NY, USA, 1–3 August 2018; pp. 579–587. [CrossRef]

34. Xu, X.; Hu, H.; Liu, Y.; Zhang, H.; Chang, D. An Adaptive IP Hopping Approach for Moving Target Defense Using a Light-Weight CNN Detector. *Secur. Commun. Netw.* **2021**, *2021*, 8848473. [CrossRef]
35. Gudla, C.; Sung, A.H. Moving Target Defense Discrete Host Address Mutation and Analysis in SDN. In Proceedings of the 2020 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 16–18 December 2020; pp. 55–61. [CrossRef]
36. Sun, J.; Sun, K. DESIR: Decoy-enhanced seamless IP randomization. In Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016; pp. 1–9. [CrossRef]
37. Achleitner, S.; La Porta, T.; McDaniel, P.; Sugrim, S.; Krishnamurthy, S.V.; Chadha, R. Cyber Deception: Virtual Networks to Defend Insider Reconnaissance. In Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats, MIST '16, Vienna, Austria, 28 October 2016; pp. 57–68. [CrossRef]
38. Clark, A.; Sun, K.; Bushnell, L.; Poovendran, R. A Game-Theoretic Approach to IP Address Randomization in Decoy-Based Cyber Defense. In Proceedings of the Decision and Game Theory for Security, London, UK, 4–5 November 2015; pp. 3–21.
39. Wang, K.; Chen, X.; Zhu, Y. Random domain name and address mutation (RDAM) for thwarting reconnaissance attacks. *PLoS ONE* **2017**, *12*, e0177111. [CrossRef]
40. ONF. *TS-012: SDN Architecture. Technical Standard*; ONF—Open Networking Fundation: Palo Alto, CA, USA, 2013.
41. ONF. *TR-521: OpenFlow Switch Specification*; Technical Reference; ONF—Open Networking Fundation: Palo Alto, CA, USA, 2016.
42. Lockheed Martin. The Cyber Kill Chain. Available online: https://www.lockheedmartin.com/en-us/capabilities/cyber.html (accessed on 30 October 2023).
43. Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G.; et al. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
44. ONF. *$P4_{16}$ Language Specification—Version 1.2.4*; Standard, The P4 Language Consortium: Palo Alto, CA, USA, 2023.
45. Intel. Intel Tofino. Available online: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino/products.html (accessed on 18 February 2024).
46. Intel. Intel Tofino 2. Available online: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino-2/products.html (accessed on 18 February 2024).
47. Intel. *Intel Tofino 3 Intelligent Fabric Processors*; Sepecification; Intel Corporation: Santa Clara, CA, USA, 2023.
48. AMD XILINX. NetFPGA-SUME FPGA Development Board. Available online: https://www.xilinx.com/products/boards-and-kits/1-6ogkf5.html (accessed on 18 February 2024).
49. AMD XILINX. Alveo SN1000 SmartNIC Accelerator Card. Available online: https://www.xilinx.com/products/boards-and-kits/alveo/sn1000.html (accessed on 18 February 2024).
50. Intel. Intel® FPGA PAC N3000. Available online: https://www.intel.com/content/www/us/en/products/sku/193920/intel-fpga-pac-n3000/specifications.html (accessed on 18 February 2024).
51. Contributors, M.P. Mininet—An Instant Virtual Network on your Laptop (or Other PC). Available online: https://mininet.org/ (accessed on 18 February 2024).
52. Manzanares-Lopez, P.; Muñoz Gea, J.P.; Malgosa-Sanahuja, J. Passive In-Band Network Telemetry Systems: The Potential of Programmable Data Plane on Network-Wide Telemetry. *IEEE Access* **2021**, *9*, 20391–20409. [CrossRef]
53. Robin, D.D.; Khan, J.I. Open Source Compiling for V1Model RMT Switch: Making Data Center Networking Innovation Accessible. In Proceedings of the 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC), Vancouver, WA, USA, 6–9 December 2022; pp. 133–138.
54. Kumazoe, K.; Shibata, M.; Tsuru, M. A P4 BMv2-Based Feasibility Study on a Dynamic In-Band Control Channel for SDN. In *Advances in Intelligent Networking and Collaborative Systems*; Barolli, L., Miwa, H., Eds.; Springer: Cham, Switzerland, 2022; pp. 442–451.
55. p4language. The Reference P4 Software Switch: Behavioral Model. Available online: https://github.com/p4lang/behavioral-model (accessed on 18 February 2024).
56. ITU-T. *Recommendation G.8261: Timing and Synchronization Aspects in Packet Networks*; Recommendation, ITU-T—International Telecommunication Union—Telecommunication Standardization Sector: Geneva, Switzerland, 2019.
57. Bierman, A.; Björklund, M.; Watsen, K. *RESTCONF Protocol*; RFC 8040; IETF—Internet Engineering Task Force: Wilmington, NC, USA 2017. [CrossRef]
58. Deanna Woodward. 100 G Sub-Categories of Data Center. Available online: https://copyprogramming.com/howto/serialization-and-serialization-times-in-40g-10g-and-100g-25g-ethernet (accessed on 30 October 2023).
59. Miller, K. Calculating Optical Fiber Latency. Available online: https://www.m2optics.com/blog/bid/70587/calculating-optical-fiber-latency (accessed on 30 October 2023).
60. Edgecore Networks Corporation. Ethernet Serialization and Times for Different Speeds. Available online: https://www.edge-core.com/cloud-data-center-100g/ (accessed on 30 October 2023).

61. Harkous, H.; He, M.; Jarschel, M.; Pries, R.; Mansour, E.; Kellerer, W. Performance Study of P4 Programmable Devices: Flow Scalability and Rule Update Responsiveness. In Proceedings of the 2021 IFIP Networking Conference (IFIP Networking), Helsinki, Finland, 21–24 June 2021; pp. 1–6. [CrossRef]
62. Cho, J.H.; Sharma, D.P.; Alavizadeh, H.; Yoon, S.; Ben-Asher, N.; Moore, T.J.; Kim, D.S.; Lim, H.; Nelson, F.F. Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense. *arXiv* **2019**, arXiv:1909.08092.