telecom

MDPI

Review

A Comprehensive Survey on Knowledge-Defined Networking

Patikiri Arachchige Don Shehan Nilmantha Wijesekara * and Subodha Gunawardena

Department of Electrical and Information Engineering, Faculty of Engineering, University of Ruhuna, Galle 80000, Sri Lanka; subodha@eie.ruh.ac.lk
* Correspondence: nilmantha@eie.ruh.ac.lk

Abstract: Traditional networking is hardware-based, having the control plane coupled with the data plane. Software-Defined Networking (SDN), which has a logically centralized control plane, has been introduced to increase the programmability and flexibility of networks. Knowledge-Defined Networking (KDN) is an advanced version of SDN that takes one step forward by decoupling the management plane from control logic and introducing a new plane, called a knowledge plane, decoupled from control logic for generating knowledge based on data collected from the network. KDN is the next-generation architecture for self-learning, self-organizing, and self-evolving networks with high automation and intelligence. Even though KDN was introduced about two decades ago, it had not gained much attention among researchers until recently. The reasons for delayed recognition could be due to the technology gap and difficulty in direct transformation from traditional networks to KDN. Communication networks around the globe have already begun to transform from SDNs into KDNs. Machine learning models are typically used to generate knowledge using the data collected from network devices and sensors, where the generated knowledge may be further composed to create knowledge ontologies that can be used in generating rules, where rules and/or knowledge can be provided to the control, management, and application planes for use in decision-making processes, for network monitoring and configuration, and for dynamic adjustment of network policies, respectively. Among the numerous advantages that KDN brings compared to SDN, enhanced automation and intelligence, higher flexibility, and improved security stand tall. However, KDN also has a set of challenges, such as reliance on large quantities of high-quality data, difficulty in integration with legacy networks, the high cost of upgrading to KDN, etc. In this survey, we first present an overview of the KDN architecture and then discuss each plane of the KDN in detail, such as sub-planes and interfaces, functions of each plane, existing standards and protocols, different models of the planes, etc., with respect to examples from the existing literature. Existing works are qualitatively reviewed and assessed by grouping them into categories and assessing the individual performance of the literature where possible. We further compare and contrast traditional networks and SDN against KDN. Finally, we discuss the benefits, challenges, design guidelines, and ongoing research of KDNs. Design guidelines and recommendations are provided so that identified challenges can be mitigated. Therefore, this survey is a comprehensive review of architecture, operation, applications, and existing works of knowledge-defined networks.

Keywords: intelligence; knowledge-defined networking; machine learning; ontology; software-defined networking

check for updates

Citation: Wijesekara, P.A.D.S.N.; Gunawardena, S. A Comprehensive Survey on Knowledge-Defined Networking. *Telecom* **2023**, *4*, 477–596. https://doi.org/10.3390/telecom4030025

Academic Editor: Sotirios K. Goudos

Received: 8 June 2023
Revised: 30 June 2023
Accepted: 20 July 2023
Published: 2 August 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In contrast to conventional networking, which distributes the control plane across network hardware, Software-Defined Networking (SDN) facilitates programming for networks by removing the fundamental network control logic from switches and routers to encourage conceptual network control centralization [1]. Owing to the controller's collection of network condition data, SDN offers more network knowledge and the ability to build safe pathways than typical hardware-dependent networking [2]. Infrastructure,

control, and application planes are the three planes that make up SDN. The key benefits of SDN over traditional networks are adaptability and programming capability. Numerous benefits are gained as a result of the SDN controller's network consciousness, including adaptive node transmission power reservation, improved routing, dynamic radio interface placement, etc. [3]. Additionally, SDN boosts networking services such as routing and load balancing and makes global enhancements thanks to the gathering of network statistics. It also promotes network creativity by making it possible for new protocols to be assessed and set up at cheaper rates [4]. As the control is conceptually centralized in SDN, any amount of physical devices can connect with each other via just one protocol. SDN has assisted in making it possible to perform new tasks and provide new services, such as traffic engineering, software development, virtualization of networks and automation, cloud-based service management, etc. [5]. However, SDN's dependability is seriously lacking, as the controller often acts as just one element of malfunction [6]. Additional difficulties for SDN include security flaws [7], and, due to the substantial movement of nodes and changing network layout, network tasks such as routing and control of transmission [8,9]. Additionally, SDN encounters difficulties, including integration with outdated networks that are not compatible with the OpenFlow protocol, the centralized controller's inability to independently control all traffic, the availability of only a few protocols for communication between the controller and services, etc. [10].

There are primarily three types of SDN architectures: centralized, distributed, and hybrid. In a centralized design, all logical control is centralized, and data plane nodes execute activities in line with the SDN controller's rules for traffic [11]. However, the control plane connection between the data plane elements and the centralized controller in this design has a larger delay [12]. Additionally, this design has limited extensibility and has the potential for inaccuracy when control plane connections disappear or are interrupted. Control is divided across numerous controllers in a distributed architecture, and the controllers may interact with one another for coordination and uniformity. The sole point of collapse and capacity issues seen in the centralized control architecture are avoided by this design. To determine the best paths, this design takes longer time than the centralized architecture [13]. A hybrid control architecture has been created to overcome the shortcomings of both distributed control and centralized control designs. The centralized controller in this system is able to adjust the degree of control exerted over the nodes from total to none depending on the situation, allowing it to behave as a blend of fully centralized control with a conceptually centralized control plane and fully distributed control, with a portion of the control plane split among the end devices [14].

The literature that is now available describes four types of functionally centralized control architecture for the SDN based on the organizational structure of the data and control planes: fully hierarchical, hierarchical data plane, hierarchical control plane, and standard. A fully hierarchical architecture contains tiered data and control planes that are separated into upper and lower planes [15]. The upper data plane is made up of border network forwarding devices such as wireless access points, eNodeB, and gNodeB, while the lower data plane is made up of SDN-enabled nodes. The upper control plane has an overall picture of the network and is made up of SDN controllers of the primary network. The lower control plane is responsible for controlling the subsystems of the network and is made up of SDN controllers of the edge network connected to the upper control plane by a cable network. Because multiple secondary SDN controllers may be allocated to different parts of the network, this design provides the maximum level of agility and programming abilities. The control plane in a hierarchical data plane SDN architecture is non-hierarchical, whereas the data plane is separated into lower and upper data planes [7]. The data plane in a hierarchical control plane SDN architecture is not separated into upper and lower data planes, but the control plane is [16]. The control plane is totally centralized and has the least adjustability in the standard SDN architecture, which is equivalent to the original SDN planes [17]. The standard SDN paradigm is the most generally used design, while the full hierarchical SDN structure is the most rarely utilized model by academics as a result

of its more complicated nature in terms of control and implementation, according to the study conducted in [18].

Knowledge-Defined Networking (KDN) is the concept of using information to generate knowledge using machine learning models or rule-based models and making network decisions accordingly [19]. The work in [19] presents a framework for cooperative knowledge building and sharing. Research conducted in [20] also uses the KDN concept for exploring the knowledge of risk reasoning through vehicular maneuver conflict. More recently, KDN has been used where node mobility is analyzed to measure the centrality degree of a region, and this knowledge is made accessible to nodes [21]. In research [22], sensor information from multiple nodes in a common geographical area is used to generate knowledge to recognize high-level contexts of control. Khan et al., in their research [23], have applied deep learning [24] to learn the transmission patterns of neighboring vehicles, which has resulted in fewer packet collisions. Knowledge creation, composition, and distribution to reduce data volume and cost have been proposed in work [25], which uses the concept of KDN. However, the preceding work failed to bring out a well-studied architecture for KDN, even though its applications have been studied by previous researchers. A knowledge-defined network has been proposed by some researchers as an architecture for applying Machine Learning (ML) to SDN [26]. KDN architecture uses the concept of knowledge-based networking; in other words, it uses information to generate knowledge using ML models or heuristic models. Even though the initial architecture of KDN was proposed in 2003 in research [27], it has gained attention recently due to advancements in machine learning. KDN has an additional management plane and a knowledge plane logically separated from the control plane compared to SDN, where the knowledge plane uses machine learning or heuristic model-based methods to process the information collected from the management plane and generate rules and knowledge to provide to other planes of the KDN. KDN has been utilized to improve network performance through automatic optimization of network traffic routing and load balancing based on real-time data analysis [28]. KDN converts the manually configured control plane actions in conventional SDN to a self-learning automated rule-generating control plane driven by knowledge generated from artificial intelligence [29].

The use of Machine Learning (ML) technology for KDN security has been popular in the recent past, where it has been used to detect intrusions [30], detect Distributed Denial of Service (DDoS) attacks [31], and detect anomalies [32]. ML has also been used for traffic classification [33], packet classification [34], predicting link failure [35], routing optimization [36], etc. in KDN. For knowledge-defined vehicular networks, machine learning has been used for predicting vehicle-to-infrastructure link life times [37], detecting DDoS attacks [38], trust-based routing optimization [39], etc. In [40], the authors show how machine learning, meta-heuristics, and fuzzy logic can be used to generate knowledge for knowledge-defined networks. AI/ML has been applied to the 5G network architecture to make knowledge-driven decisions for slice management, network service orchestration, vertical domain cross-layer optimization, management analytics, and anomaly detection [41]. By automating network management tasks, KDN can reduce the need for manual intervention, lower operational costs, and increase energy efficiency [42]. Work in [43] shows how the KDN concept can be used for closed-loop network monitoring to realize a self-driving network concept. Similarly, another piece of research highlights how closed loop control can be used for automatic routing in a KDN using deep reinforcement learning to learn experiential knowledge, which also includes network monitoring to realize the interaction with the environment [44]. A self-organizing routing algorithm that reactively finds the most reliable route using a deep neural network in a self-organizing knowledge-defined network has been investigated in [45]. Driven by the benefits of automation and recommendation due to the knowledge plane in KDN, a self-driving system that selects the optimal path for service function chaining and reactive traffic functioning using graph neural networks has been studied in [46]. A framework for identifying heavy-hitter flows using machine learning in KDNs has been investigated in [47]. ML has also been used

for video flow classification in 5G KDNs [48]. However, the KDN concept is still very pre-mature, and it lacks standardization and protocols for intra-plane communication [49].

Early surveys were totally focused on SDN architecture and applications without any review of intelligent networking [50]. More recent surveys have reviewed the concept of knowledge-based networking by categorizing machine learning applications in SDN and identifying challenges in applying machine learning in SDN [51,52]. The survey carried out in [53] collects and reviews machine learning-based SDN solutions that emphasize machine learning-based solutions, evaluation parameters, and evaluation environments. More closely related to our survey is the survey carried out in [54], which provides an overview of the KDN architecture. However, the previously mentioned work discussed more on machine learning-based applications applicable in KDN and challenges associated with those applications rather than reviewing on the whole paradigm itself. However, our survey comprehensively reviews existing protocols, languages, standards, interfaces, models, and functions related to each plane of the KDN architecture and, finally, discusses the benefits and challenges, providing design guidelines and recommendations, thus providing a complete tutorial on the knowledge-defined networking concept to the reader, with reference to the existing literature that is analyzed and discussed. The hierarchical organization of this survey is graphically illustrated in Figure 1.

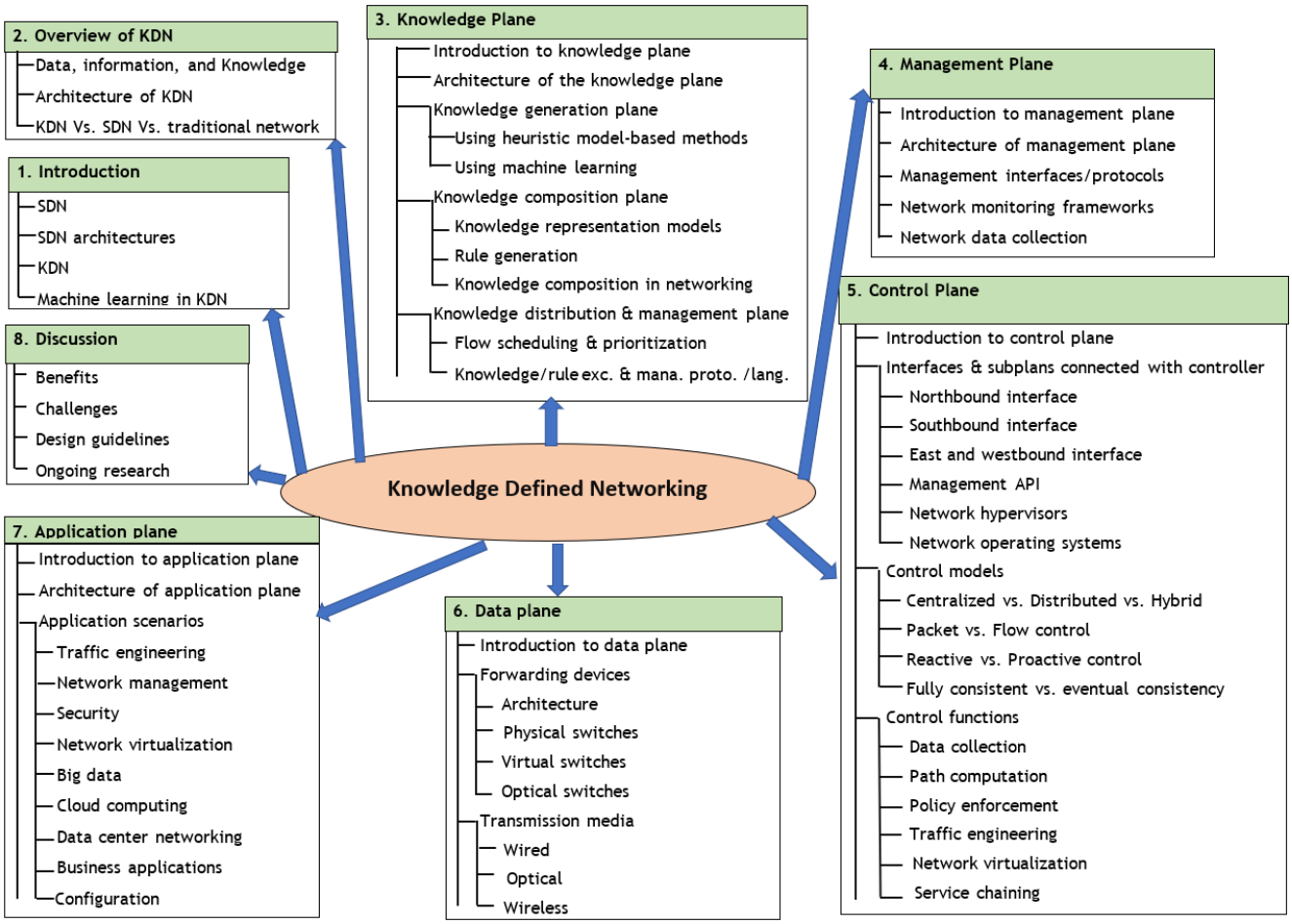

Figure 1. Hierarchical organization of the KDN survey.

As evident from Figure 1, the rest of the paper is organized as follows: Section 2 presents a brief introduction to data, information, and knowledge (Section 2.1), then presents an overview of the KDN architecture (Section 2.2), and compares it with SDN and traditional networks (Section 2.3). Section 3 presents an introduction to knowledge plane (Section 3.1); architecture of the knowledge plane (Section 3.2); functioning of the

knowledge generation plane (the plane where knowledge generation occurs using heuristic model-based methods and different machine learning techniques) is discussed in detail with reference to the existing literature (Section 3.3); knowledge representation models, rule generation techniques, and knowledge composition examples from the existing literature are discussed in the knowledge composition plane (Section 3.4); and existing knowledge and rule management and exchange protocols/languages, flow scheduling, and prioritization approaches for knowledge and rule distribution are discussed in the knowledge management and distribution plane (Section 3.5) in detail. Section 4 presents the management plane, explaining its function (Section 4.1), architecture (Section 4.2), management interfaces/protocols (Section 4.3), network-monitoring frameworks (Section 4.4), and data collection strategies and types of data collected (Section 4.5). Section 5 presents an introduction to the control plane (Section 5.1), explains sub-planes and interfaces with protocols and standardization (Section 5.2), describes control models (Section 5.3), and describes control functions (Section 5.4). Section 6 presents the data plane, first with an introduction to the data plane (Section 6.1), then presents the architecture and operation of forwarding devices (Section 6.2), and discusses different transmission media used in the infrastructure plane (Section 6.3). Section 7 presents the application plane first with an introduction to the application plane (Section 7.1), then the architecture of the application plane (Section 7.2), and, finally, different application scenarios with respect to existing literature are discussed (Section 7.3). Section 8 discusses the benefits (Section 8.1), challenges (Section 8.2), design guidelines (Section 8.3), and ongoing research (Section 8.4) of KDN in detail. Finally, Section 9 concludes the paper with recommendations and future research.

1.1. Objectives and Key Issues Addressed

The objective of this research is to educate the reader comprehensively on the knowledge-defined networking paradigm. First, the reader is provided with an overview of the KDN framework, and then each plane of the paradigm is discussed, such as standards, protocols, models, interfaces, functions, applications, etc., with reference to the existing literature. Another key objective of the survey is to study the benefits and challenges of the KDN framework in order to provide design guidelines and recommendations that will encourage academicians to perform more research on KDN. Finally, this survey qualitatively analyzes the existing works on different aspects of KDN by grouping and stating characteristics and analyzing the individual performance of the solutions.

1.2. Contributions to the Existing Literature

- We are the first to review on knowledge-defined networking, which will provide a useful reference for future researchers who investigate more in this area;
- We compare traditional networking with SDN and KDN;
- Each plane of the KDN architecture is discussed in detail with reference to the existing literature;
- Benefits, challenges, design guidelines, and ongoing research on the KDN architecture are discussed.

1.3. Research Methodology

This survey is a qualitative research study that critically analyzes and interprets existing research work on knowledge-defined networking. The population of the survey is comprised of all original research works and web articles published related to KDN and artificial intelligence/machine learning/knowledge-based SDN. Out of that population, we sampled 613 of the most relevant research works published from 1980 to 2023 related to architecture, operation, applications, interfaces, functions, languages, and protocols of each plane of the KDN paradigm by searching scientific databases. We used IEEE Xplore, the ACM Digital Library, ScienceDirect, MDPI, and Google Scholar as searching databases. However, after careful analysis, we identified that 43 works presented very similar content to one or more works already in the sample. Therefore, we removed these

43 redundant references, which provided redundant concepts, to finally reach 570 original research references in the cleaned sample. Later, we added five survey articles published on knowledge-/machine learning-based SDN in order to compare our review with existing reviews, thus increasing the total references to 575. Thus, the approach of the survey was longitudinal. The research works were selected based on their relevance to the content discussed in the survey. We used tabular data structures to analyze research qualitatively by grouping them into categories and analyzing the characteristics of the frameworks. We sampled research works based on relevancy without being biased by any publisher or time of publication. However, priority was given to journal publications over conference publications to improve the validity and reliability of the survey. Ethical considerations are not applicable, as this is a survey in the computer networking domain.

2. Overview of Knowledge-Defined Networking (KDN)

2.1. A Nutshell on Data, Information, and Knowledge

A piece of data is defined as an atomic value with a unit, which is the most fundamental element that is unprocessed and raw and contributes to knowledge generation. Let us understand these concepts with reference to an example from a vehicular network. An example of data can be the Global Position System (GPS) position $(x1, y1, z1)$.

Information is a processed and organized collection of data that is more meaningful than raw data and can be used to make a decision. An example of information can be the distance from the controller to a vehicle, which is calculated by processing data containing the status of the vehicle, such as the GPS position of a given vehicle at a given timestamp $(vehicle_1, 15{:}05, (x1, y1, z1))$. In conventional networks such as Vehicular Ad Hoc Networks (VANETs), information such as safety notifications, vehicle state information, sensor measurement information, navigation information, etc. is exchanged between the vehicles. Conventional vehicular networks use this exchanged information to make decisions on network functions such as routing without generating knowledge [55].

On the other hand, in KDNs, the approach is shifted from the information-centric approach to the knowledge-centric approach. Knowledge is defined as the state of understanding obtained through experience, learning, and the analysis of collected data/information. In other words, knowledge is an abstract content obtained by learning and analyzing a large amount of data/information [56]. Thus, the decision-making power of knowledge is much higher than that of information. An example of the knowledge that can be obtained using a set of previously mentioned information could be whether $vehicle_1$ is likely to have a collision or not, which can be inferred by understanding and learning a lot of information, such as safety notifications, sensor measurement information, navigation information, etc., about the given vehicle and other vehicles in the neighborhood.

2.2. KDN System Architecture

Figure 2 shows the high-level block diagram of the KDN architecture [27,57].

As evident from Figure 2, a KDN consists of five main planes, which are briefly introduced below.

- Knowledge Plane—The knowledge plane consists of three sub-planes. The knowledge generation plane generates knowledge using data/information, either by using heuristic model-based methods or machine learning methods. In the knowledge composition plane, generated knowledge and universal knowledge are composed using an ontology editor to produce composed knowledge, which can be used to generate rules by orchestration with user intent. The knowledge distribution and management plane is responsible for storing knowledge and rules using a knowledge base and performing the inserting, updating, deletion, and exchange of knowledge and rules using languages and protocols [25];

- Management Plane—The management plane (also referred to as the measurement plane) operates in parallel with the KDN controller and is responsible for collecting the processes and data/information from the network devices, monitoring the network device status, and configuring the network device. Note that, in KDN, the management plane can be influenced by the application plane to implement configuration policies and by the knowledge plane to aid in real-time network monitoring, whose monitoring output can be used to dynamically configure the network [57,58];
- Data plane—The data plane is composed of forwarding devices that can store, forward, or process data according to the flow rules sent by the control plane. In KDN, the data plane is required to send data requested by the management and control planes [59];
- Control plane—The control plane consists of one or more SDN controllers based on the architecture and is responsible for sending flow rules, access control rules, QoS-based traffic prioritization rules, etc. to the data plane. Control in KDN is driven by both application policies and real-time rules or knowledge generated from the knowledge plane [60];
- Application plane—This plane provides a platform for network applications to communicate requirements to the underlying network infrastructure. It also enables network administrators to define network policies specific to applications and define network configuration policies that are more aligned with high-level business needs and objectives in a centralized manner, where the application logic is decoupled from hardware. In KDN, application policies can be dynamically updated based on knowledge [61].

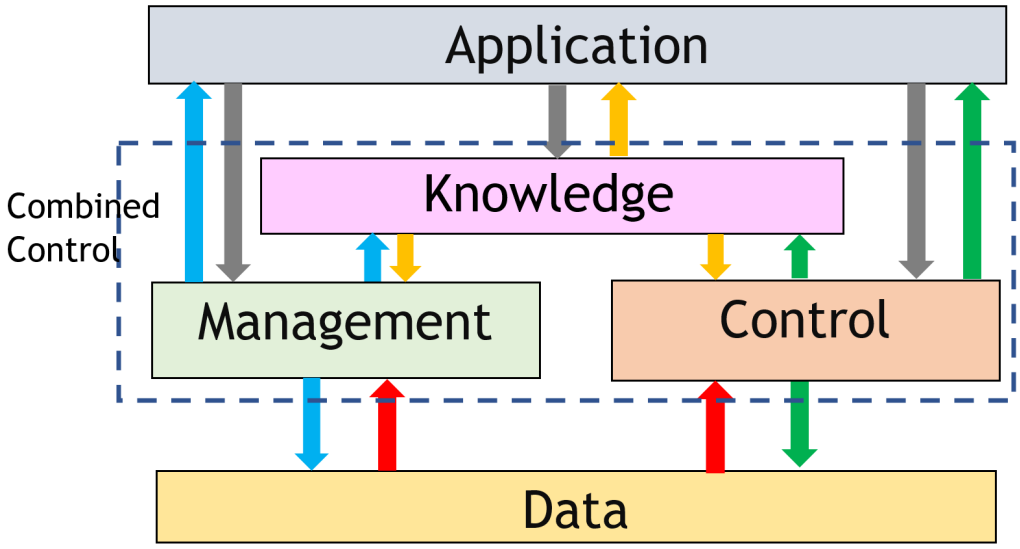

Figure 2. High-level block diagram of the KDN architecture.

Note that knowledge, management, and control planes in KDN can be abstracted to a combined control plane, as shown in Figure 2, where the combined control plane's task is to manage the network and make control decisions driven by knowledge and application policies [62]. The abstracted combined control plane in KDN and the logically centralized control plane in SDN have exactly the same tasks of network management and network control. However, the abstracted combined control plane differs with respect to the control plane in SDN by the fact that control actions in KDN are self-learning, adaptive, and knowledge-driven, while the control plane in SDN is driven by application policies, since SDN does not have a knowledge plane.

2.3. KDN vs. SDN vs. Conventional Network

Traditional networking is the oldest approach to networking and involves manual configuration and management of devices. This method of networking has been prevalent since the beginning of networking, and it still prevails in modern communication networks.

SDN is a more recent approach that separates the control plane from the data plane and enables greater flexibility in network design. The control plane is moved to a centralized location, and network administrators use software applications to manage the network. This paradigm allows network administrators to manage networks more easily and quickly due to increased flexibility and programmability.

KDN creates a self-learning, self-optimizing, and self-healing network by integrating AI and ML technology. Data analysis is a tool used by KDN systems to automatically improve network performance, adapt to shifting network circumstances, and spot and resolve potential network issues before they become bigger difficulties.

2.3.1. Decoupling of Logical Planes

In traditional networks, the data plane and control plane are tightly coupled, and devices such as routers perform both data processing and control functions. Thus, traditional networks are difficult to manage and automate.

In SDN, the main purpose is to decouple the data and control planes logically, which allows network administrators to manage and program networks using software rather than hardware. The logically decoupled controllers communicate with forwarding elements using protocols such as OpenFlow, ForCES, etc. However, in SDN, network management is embedded in the control plane, and there is no knowledge plane to generate knowledge. Thus, SDN does not emphasize knowledge generation in arriving at control decisions.

Note that KDN architecture is an extension of the originally proposed SDN architecture by decoupling the management plane from control logic and introducing a new logically decoupled knowledge plane. KDN emphasizes knowledge representation, reasoning, and decision-making to manage, configure, and control networks. It uses domain-specific knowledge representation (knowledge ontologies) to automate the management of network devices and create an intelligent network that can learn and adapt to changing conditions.

Thus, the key differences in KDN with respect to SDN lie in the logical decoupling of the management plane from control logic and the introduction of a new logically decoupled knowledge plane.

2.3.2. Network Programmability

In traditional networks, each network device is required to be manually configured, which can become infeasible in large networks due to time consumption and the error-prone nature.

Even though both SDN and KDN aim to improve network programmability, they differ in the approach by which they implement it. SDN uses application programming interfaces and software controllers to program network behavior, while KDN additionally involves knowledge-based systems such as AI and machine learning in addition to application of programming interfaces and software controllers to automate network control and management.

2.3.3. Control Plane

The control plane is dispersed among network devices in conventional networking. Despite the logical centralization of the control plane in SDN, the controller's decisions are not guided by knowledge but rather by the policies of the network applications. In KDN, the control plane is both logically centralized and self-learning, as control is based on both network policies and knowledge generated by AI and machine learning techniques.

2.3.4. Management Plane

In traditional networking, networking devices are managed and configured by the network administrator using a distributed approach. In SDN, management is centralized; however, it is embedded in the control plane and not decoupled as a separate plane. In KDN, management functions are decoupled from control logic, and separate protocols are used for data collection, network monitoring, and network configuration, which makes troubleshooting in case of failure much easier than in SDN. Furthermore, network management can be automated using knowledge learned from the network to update management policies in the application plane to automatically adapt to network changes in real-time with minimum human intervention.

2.3.5. Knowledge Plane

In traditional networking, decisions are made not based on knowledge at all. Such networks are based on strict rules or policies enforced by network administrators. In SDN, knowledge is also not generated; however, the centralized controller uses information collected from the network, such as device statistics, status, etc., to develop a global view of the network to help in arriving at control decisions based on policies enforced by applications. In KDN, based on data collected from the network, machine learning or a heuristic model-based approach will be used to generate knowledge, which will be used in deriving rules with the aid of knowledge composition or by providing knowledge to the control plane to use in arriving at control decisions. Furthermore, the knowledge/rules can be provided to the application plane to update policies and to the management plane to aid in network monitoring.

2.3.6. Application Plane

In conventional networking, services are provided depending on the capabilities of underlying devices, and the application plane is closely tied to the network infrastructure. This can limit the flexibility of the network and make it challenging to adapt to shifting requirements. Instead of relying on the network infrastructure's capabilities, SDN offers more flexibility and enables services to be supplied in accordance with application needs. In SDN, the application plane is decoupled from the underlying physical network and is dynamically defined and reconfigured according to the requirements of the service. KDN goes a step farther in terms of the application plane's versatility. In KDN, the application is continually optimized depending on the service's requirements and network performance, in addition to being developed dynamically. This strategy enables the more effective use of network resources and may result in better service delivery.

2.3.7. Network Architecture

Traditional networks have a static network architecture, while SDN has a flexible network. In KDN, flexibility is even greater than in SDN, as it is a self-learning network that can adapt to changing conditions in real time.

2.3.8. Operational Cost

Traditional networking is expensive to operate, as it requires skilled programmers to configure and manage the network. Using a logically centralized control plane, the operational cost of SDN can be drastically reduced compared to traditional networks. The operational cost savings in KDN are even higher than in SDN, as it uses AI and machine learning techniques to generate and compose knowledge that can be used to automate network management tasks with the least involvement of humans (network administrators).

2.3.9. Security Features

Traditional networks have limited security features, which are basically implemented using firewalls and intrusion detection systems. SDN provides better security compared to traditional networks due to the centralization of security policies and easier deployment of security protocols, even though SDN is known for different attacks such as denial of service attacks, malware attacks, spoofing attacks, data modification attacks, etc. KDN provides even more security than SDN by using ML to detect threats in real time using the knowledge plane and mitigate the threats using the control plane.

Table 1 summarizes a comparison among KDN, SDN, and traditional networking.

Table 1. Comparison of KDN, SDN, and traditional networks.

Parameter/Feature	Traditional Network	SDN	KDN
Logical planes	Data, control, and application coupled	Data, control, and application decoupled	Data, control, management, knowledge, and application decoupled
Network programmability	Manual and complex	Manual and simple	Hybrid and simple
Control plane	Distributed across devices	Logically centralized and driven by application policies	Logically centralized and self learning driven by application policies and knowledge
Management plane	Manual and distributed	Manual and centralized coupled with control plane	Hybrid and centralized decoupled from control plane
Knowledge plane	Totally knowledge ignorant	Knowledge ignorant. However, make use of global network information in arriving at decisions	Generate knowledge, compose knowledge, generate rules, store and disseminate them to other planes
Application plane	Tightly coupled with infrastructure and inflexible	Decoupled with infrastructure and can be redefined dynamically based on service needs	Decoupled from infrastructure, can be redefined and optimized based on service needs and network performance
Network architecture	Static	Flexible	Flexible, self learning, and automated
Operational cost	High	Medium	Low
Implementation cost	Low	Medium	High
Security features	Low	Medium	High

3. Knowledge Plane

3.1. Introduction to Knowledge Plane

The knowledge plane is a logical layer that is responsible for generating, processing, and disseminating knowledge about the network. The knowledge plane helps network operators improve their ability to manage network behavior and adapt to changing conditions. The knowledge plane includes various types of knowledge, such as declarative knowledge (rules), procedural knowledge (processes), and contextual knowledge (knowledge about the environment and the conditions in which the network operates) [63]. The data for generating knowledge can come from various sources, such as network devices, sensors, etc., which are analyzed to generate insights and enable intelligent decision-making. By making full use of the knowledge plane, network operators can proactively identify and address issues before they become problems, optimize network behavior to meet changing demands, and implement advanced security measures to protect against threats.

Knowledge representation/modeling languages can be used to represent and manipulate knowledge in a machine-readable format. They provide the data model and syntax for representing the knowledge. Resource Description Framework (RDF), Web Ontology Language (OWL), Ontology Inference Layer (OIL), RDF Schema (RDFS), and Knowledge Interchange Format (KIF) are the main knowledge representation languages that have been proposed to be utilized in knowledge-defined networks. These knowledge representations are discussed in detail in subsequent sections.

3.2. Architecture of the Knowledge Plane

The architecture of the knowledge plane containing the three knowledge sub-planes and its communication with the network [25] are graphically illustrated in Figure 3.

As evident from Figure 3, there are three sub-planes in the knowledge plane, which are listed below:

- Knowledge generation plane—Generates descriptive knowledge from raw data using a knowledge generation model;
- Knowledge composition plane—Identifies the relationship between different pieces of knowledge to compose knowledge using an ontology editor, which can be used to generate rules considering user intent;
- Knowledge distribution and management plane—Manages and exchanges raw data (within the knowledge plane), descriptive knowledge, rules, and control messages between the knowledge plane and other planes.

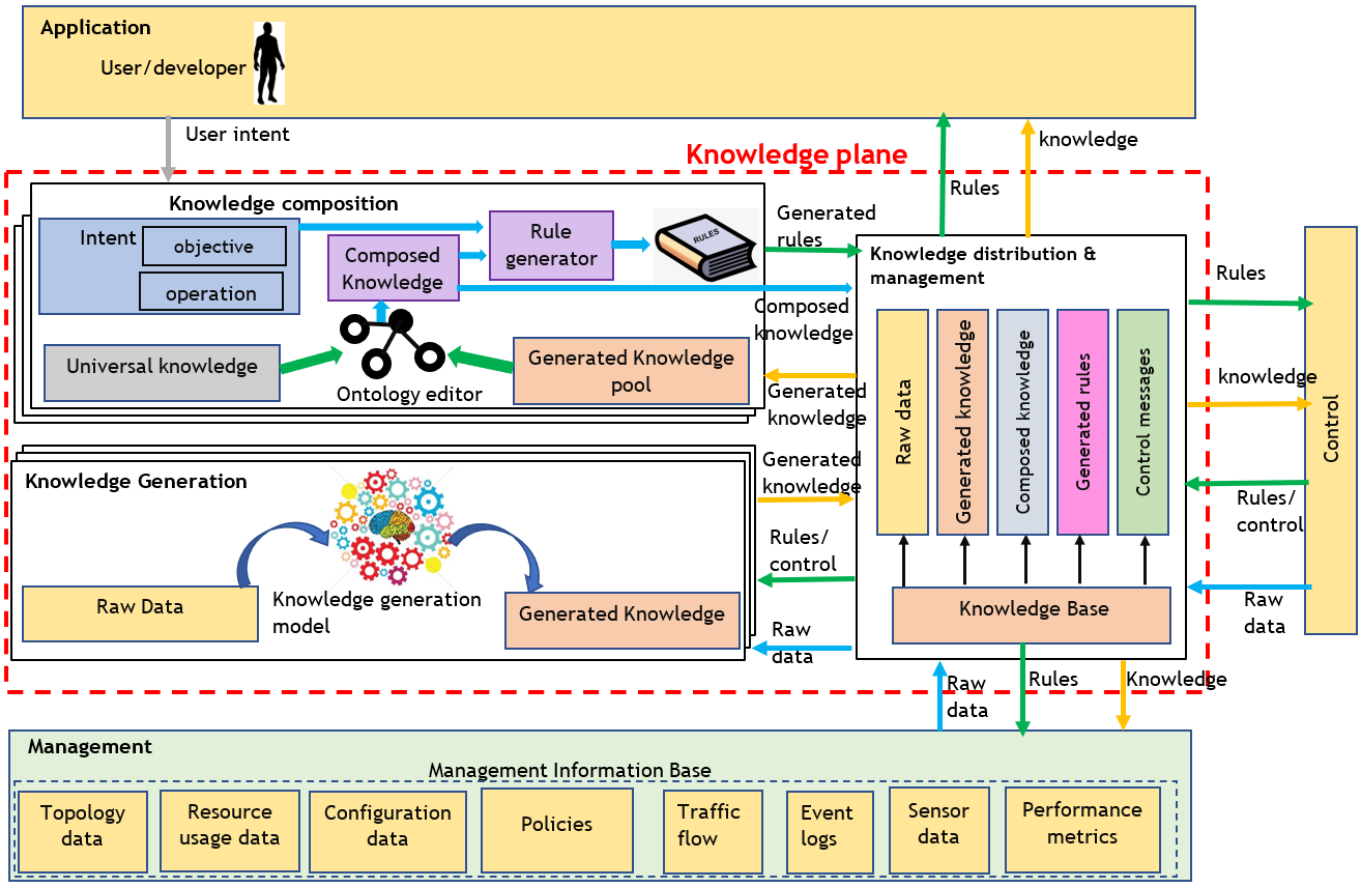

Figure 3. Architecture of the knowledge plane and its interaction with the network.

The knowledge and rules generated from the knowledge plane can be helpful for fault diagnosis and mitigation, automatic reconfiguration, intrusion detection, etc. It is a requirement of the knowledge plane to operate successfully in the presence of limited and uncertain inputs [27]. The knowledge plane architecture given in Figure 3 is described in detail in the following sections.

3.3. Knowledge Generation Plane

The knowledge generation plane extracts descriptive knowledge from raw data using a knowledge generation model that uses a learning technique that can be either a heuristic model-based method or a machine learning method. Knowledge generation effectively reduces the volume of information transmitted to the controller, as the descriptive knowledge produced from a large volume of data is much smaller. Different types of data will be

collected by the management and control planes, as is discussed in Sections 4.5.2 and 5.4.1, respectively, and can be fed as input for knowledge generation. For example, the input data to the knowledge creation plane can be the data traffic received at the controller, whereas the output knowledge can be the traffic class.

The generated knowledge can be modeled using the Resource Description Framework (RDF) knowledge representation language. RDF represents knowledge as a set of triplets, which consist of a subject, predicate, and object. In RDF, subjects and objects are identified by Uniform Resource Identifiers (URIs), and predicates represent relationships between them. This structure makes RDF a flexible and extensible knowledge representation model that can represent diverse types of knowledge [64].

3.3.1. Generating Knowledge Using Heuristic Model-Based Methods

Heuristic model-based methods usually involve using a mathematical model to describe raw data [65] and using the inner correlation of the data [66] to generate knowledge. Some use simple heuristic methods such as first-order logic, fuzzy logic, Markov logic, data fusion, etc. to generate knowledge [67]. However, this type of technique may need to use numerical analysis techniques to recover from missing values for the historical data.

In the literature on network communication, there are many examples of using heuristic model-based methods to generate knowledge. Packet classification is the core of traffic classification techniques that categorize packets with a traffic descriptor or with user-defined criteria. The Aggregated Bit Vector (ABV) algorithm has been implemented in parallel using Graphic Processing Units (GPUs) for packet classification in networks [68]. Some have used a Binary Content Addressable Memory (BCAM) scheme and a lookup mechanism for packet classification, which has yielded lower memory consumption at the expense of low throughput [69]. A new data structure known as the Range Query-Recursive Model Index (RQ-RMI) has been used for training an algorithm for packet classification, which can be considered as a computational approach [70]. A practical multi-tuple packet classification algorithm called Dynamic Discrete Bit Selection (DDBS), which employs dynamic heuristic schemes at the bit level in order to explore the characteristics of the classification rule sets, has been conducted in [71], which has a high classification speed with a lower storage requirement. Some have used cross-product and linear search for packet classification, which have high feasibility and scalability [72]. A Tuple Space Search (TSS)-assisted algorithm has been used in software-based switches (Open v) in KDN for packet classification [73]. The TSS-based approach is a two-stage framework where partial decision trees are constructed from rule subsets grouped with respect to small fields in the first stage, and, in the second stage, a TSS-based algorithm is used to classify subsets following tree construction. Interval-valued fuzzy logic has been used to classify video streams under varying network conditions that affect the behavior of network flow [74].

Model-based heuristic algorithms have been used for generating knowledge in the form of routing in KDN architectures. Early KDNs have used the extended Dijkstra shortest path algorithm for generating routes [75]. An optimization-based segment routing technique that uses Multi-Objective Particle Swarm Optimization (MOPSO), which is a technique that divides the end-to-end path into segments, aiming to minimize the cost of each segment, has been used for KDNs [76]. Some have attempted to optimize delay, blocking probability, and network utilization when finding routes using an adaptive greedy flow routing algorithm [77]. Another research work investigates on a dynamic and adaptive multi-path routing algorithm that computes routes with constraints on packet loss, time delay, and bandwidth, resulting in higher QoS for multimedia applications in KDN [78]. Similar to the preceding work, researchers have attempted to select an optimum path satisfying video QoS parameters for routing using an algorithm [79]. Some use ant colony optimization for routing in KDN to select the optimum path from multiple paths [80]. Recently, an integer programming problem for maximizing the ratio for energy saving in KDN that quantifies energy efficiency based on link utility intervals has shown good results as an energy-efficient routing scheme [81].

Intrusion detection and security systems in KDNs have used heuristic model-based methods to achieve that task. An optimization framework that optimally selects the sampling rate for each switch to sample traffic flows for inspection of malicious network traffic in large networks is presented in [82]. An information security management system and an intrusion detection system with a fuzzy logic-based decision-making module have been used in KDNs for making decisions regarding security [83]. A flow-based and packet-based intrusion detection system known as Kangaroo intrusion detection, which uses consecutive jumps, similar to a kangaroo, for announcing the attacks to the KDN controller, and other IDSs that use an attack detection algorithm to detect attacks from packets and flows, have been utilized in work [84]. A meta-heuristic approach to detecting DDoS attacks in KDNs using the Lion optimization technique, which is robust enough to detect the DDoS attack with the least magnitude of attack traffic, has been proposed [85]. An anomaly-based intrusion detection system using fuzzy logic combined with an InfoGain feature selection method has shown promising results for the detection of DDoS attacks [86]. An evolving fuzzy system to discriminate anomalies (normal and attack network situations) has been used to classify network traffic as normal or attack, by capturing time-series data, which are analyzed to establish a model of the normal network situation that evolves over time [87].

Furthermore, heuristic algorithms have been used for controller placement problems. Evolutionary algorithms have been utilized to solve a large-scale multi-objective controller placement problem, where capacities of the controllers and loads of switches are considered as constraints, and latency between nodes and controllers, the latency among controllers, and load balancing are considered as objectives [88]. A heuristic algorithm that computes the controller placements with at least the required reliability to yield a fault-tolerant placement that focuses on the number of controllers, number of nodes under the controller, and position of the controller has been studied in [89]. Some have considered energy consumption minimization in placing the controller, which uses a binary Integer Linear Programming optimization (BILP) problem, having constraints as the delay of the control path and the load of the controller. However, due to the computational complexity of BILP, a suboptimal solution using a genetic heuristic algorithm for energy-aware controller placement that yields results close to the BILP solution has been proposed for large networks [90]. Another controller placement approach uses particle swarm optimization and the Firefly algorithm to place the controllers, considering the effects of controller-to-switch latency, inter-controller latency, and multi-path connectivity between the switch and the controller [91].

3.3.2. Generating Knowledge Using Machine Learning Methods

The input data or information may be utilized to extract features using machine learning algorithms. Before putting raw data into machine learning models, high-level features are typically extracted to create information in order to increase the accuracy of the models [92]. However, for knowledge creation using ML techniques, high computational resources are required. For example, for autonomous driving applications, frames of images from multiple cameras or sensors are required to be input, which requires high computational power in knowledge generation. In such cases, distributed knowledge generation models can be used to divide the knowledge generation workload among multiple machine learning models, where the knowledge generated by each model can be later combined using the knowledge composition plane. Some research suggests that Artificial Intelligence (AI)-based approaches yield better results for knowledge generation than algorithmic approaches [27]. Five approaches to machine learning are available in order to generate knowledge from the data/information in communication networks:

- Supervised learning—In supervised learning, a training data set contains labeled inputs with corresponding labeled outputs for the supervised machine learning algorithm for training (fitting). During training, the ML algorithm learns the underlying patterns in the input data to retrieve the outputs. Supervised learning involves either a classification or regression problem. A classification problem has discrete output

variables producing qualitative outputs, whereas a regression problem has continuous output variables producing quantitative outputs [93];

- Unsupervised learning—Unsupervised learning develops knowledge by choosing a cluster of related items from the provided input data set (no supervision is required). It does not train using a labeled data set [94];
- Semi-supervised learning—In this method of learning, both labeled and unlabeled datasets are used to train the machine learning model [95];
- Integration of supervised with supervised, unsupervised, or semi-supervised learning approaches—In this integrated method, at least two machine learning models are used to complete a specific job [96];
- Reinforcement learning—With reinforcement learning, an agent in a given state acts to maximize potential rewards from the surrounding environment in the future [97].

Now, let us review each of the approaches to machine learning for knowledge generation in knowledge-defined networks.

Supervised Learning Approaches for Generating Knowledge

Deep Neural Networks (DNN)

An Artificial Neural Network (ANN) is a collection of node layers, comprising an input layer, an output layer, and optional one or more hidden layers in which each node (neuron) is associated with a weight and a threshold value [98]. A Deep Neural Network (DNN) is an ANN that necessarily consists of one or more hidden layers between the input layer and the output layer of the ANN [99]. The weights and biases of the neurons are adjusted to minimize a user-defined loss function during DNN training in order to find a mathematical relationship between the inputs and the outputs [100]. Deep learning has been used for detecting network intrusions and for flow-based anomaly detection in the centralized controller, where deep learning has resulted in better performance (except for accuracy) compared to an approach using a Recurrent Neural Network (RNN) [101]. A self-organizing routing algorithm that reactively finds the most reliable route using a deep neural network in a self-organizing knowledge-defined network has been investigated in [45]. A Convolutional Neural Network (CNN) is an ANN that has at least one convolution layer that performs the mathematical operation convolution instead of matrix multiplication [102]. Work in [103] uses a CNN and Continuous Wavelet Transform (CWT) to detect DDoS attacks in KDNs by differentiating attack samples from normal traffic. A Rectified Linear Unit (ReLU) DNN that can handle large databases of classification rules is employed in [104] for packet classification in large networks. Some have used DNNs for traffic classification and prediction in KDNs in order to process the huge amount of data received by the centralized controller [105]. An energy efficiency optimization framework based on traffic prediction in knowledge-defined networking using a DNN having a Gated Recurrent Unit (GRU) layer capturing the temporal characteristics of the network traffic, aimed at reducing network energy consumption while ensuring communication quality, is presented in [106]. Flow-based anomaly detection has been used in intrusion detection for a KDN system using a GRU–Long Short-Term Memory (LSTM)-based DNN [107]. Deep learning has also been utilized to identify the presence of link failures in a complex multi-route optical network by analyzing a large amount of data received from the optical network [108]. A link failure at handover mitigation scheme by continuously observing and tracking signal conditions using an LSTM-based DNN, where the behavior of these signal conditions is taken as input to the neural network, which acts as a classifier, classifying the event in either handover failure or success in advance, is used in [109]. Thus, in the preceding scheme, by advancing the prediction of handover status, handover failures can be mitigated. DNNs have also been used to replace heuristic routing optimization models such as mixed integer linear programming problems, which are trained on optimal decisions using flows from known traffic demands [110]. Due to facing of frequent communication interruptions and poor stability inherent in 3D Flying Ad Hoc Networks (FANETs), a DNN-based routing consisting of a 3D two-space division and DNN-based forwarding that yields better performance in terms of packet delivery rate compared to

conventional routing protocols is presented in research [111]. Wijesekara et al. generate knowledge regarding link lifetimes and one-hop channel delays related to wired and wireless communication channels in a heterogeneous knowledge-defined vehicular network using DNNs, in order to utilize the generated knowledge in a hybrid stable distance and stable delay based OpenFlow compatible adaptive routing algorithm which yields better routing performance in terms of packet delivery ratio, latency, and communication cost [112]. A DNN-based routing framework called "NeuRoute", which predicts a traffic matrix in real-time using a DNN and generates forwarding rules to optimize network throughput for KDN, is presented in [113]. A temporal-aware Quality of Service (QoS) prediction using a DNN with GRUs with the aid of feature integration has been employed in [114]. In [115], the authors present a DNN model for multiple attribute QoS prediction, where multi-task prediction is achieved by stacking task-specific perception layers on the shared neural layers.

Decision Trees

A decision tree consists of nodes that represent decisions or attributes of data, and the branches represent the possible outcomes or class labels that result from each decision. In a decision tree, any path beginning from the root is described by a data-separating sequence until a Boolean outcome at the leaf node is achieved. A creation of a decision tree involves the recursive splitting of data based on the most informative features until a stopping criteria is met [116]. Decision trees have been used for the detection of Distributed Denial of Service (DDoS) attacks in KDNs [117]. A DDoS attack is a threat that is used by cybercriminals to deny the service of a network resource, such as the controller in the KDN paradigm, by making it unavailable to its intended users. Furthermore, decision trees have been extensively used for packet classification in computer networks using different cutting approaches such as Cutsplit, which exploits the benefits of cutting and splitting techniques adaptively while having a low memory consumption [118]; Bitcuts, which has fast tree traversal speed using bit-level cutting [119]; and Hypercuts, which is a decision tree in which each node in the tree represents a k-dimensional hypercube, which uses an extra degree of freedom and a new set of heuristics to find optimal hypercubes for a given amount of storage [120]. Another frequently used classification task in decision trees is network traffic classification. In [121], traffic classification using a decision tree is used for vulnerability detection in the supervisory control and data acquisition network of a smart factory. Some have used the C4.5 decision tree algorithm along with an entropy minimum description length discretization algorithm for traffic classification, whose performance has been enhanced by implementing it in a Field Programmable Gate Array (FPGA) and multicore platforms [122]. An improved decision tree-based algorithm for traffic classification that uses C4.5 decision trees in the Hadoop platform called HAC4.5, which parallelizes the decision tree algorithm using the Hadoop platform to classify big data traffic with higher accuracy and low computational time, is presented in work [123].

Support Vector Machines (SVMs)

A support vector machine is a supervised machine learning approach used for classification and regression problems where an optimal decision boundary (hyperplane) is used to differentiate the classes or for regression [124]. A multiclass classification support vector machine has been used to detect two types of flooding-based DDoS attacks with the minimum disturbance to legitimate users in a KDN [125]. Another research project uses hierarchical flow- and packet-based anomaly detection using support vector machines as an intrusion detection system in KDN. In this approach, the first level is a flow-based IDS, which passes the flow to a packet-based IDS for further actions if the flow-based IDS detects an attack [126]. Data traffic based on the application has been classified in a KDN using an SVM, where the classification accuracy has been poor compared to the Naive Bayes machine learning model [127]. Support vector machine-based application traffic identification and classification such as Youtube, Facebook, etc. has been achieved in a KDN scenario with high classification accuracy [128]. Some have classified packets based on the action/flow of each packet in KDNs using support vector machines, where five

features of the IP header have been extracted before classification [129]. In a study that investigated the performance of the SVM supervised machine learning model and the K-means clustering unsupervised machine learning model for traffic classification, SVM yielded a higher classification accuracy than K-means [130].

Random Forest

A random forest consists of many decision trees, where each decision tree provides its prediction such that the class with the highest count becomes the classification of the model [131]. Random forest models have been used for classification, feature selection, and proximity metrics for behavior-based intrusion detection in KDNs [132]. The purpose of an intrusion detection system is to capture any malignant activities in the network before they cause severe damage to the network. The DDoS attack on the controller of KDN, which leads to resource exhaustion and the non-reachability of the services given by the controller, has been detected using random forests. However, the random forest classifier has yielded inferior performance to the SVM in detecting a DDoS attack [133]. A random forest has yielded a higher classification accuracy and a lower false positive rate for detecting the presence of an attack in a network intrusion detection system compared to neural networks and Naive Bayes machine learning classifiers [134]. Internet protocol traffic classification using the flow statistics collected by the controller and Transmission Control Protocol (TCP) flow in KDN has been achieved using a random forest classifier, which yielded a higher accuracy compared to gradient boosting classifiers [135]. Statistical learning, which involves estimating application end-to-end quality of service metrics such as frame rate and response time from device statistics, has been used in an OpenFlow network using a random forest machine learning classifier, which produced a better classification performance than the regression tree [136].

K-Nearest Neighbor

K-nearest neighbor is a supervised machine learning algorithm that groups data into coherent clusters, based on the assumption that similar points exist close to one another, and classifies the newly input data based on the similarity of previously trained data [137]. The K-nearest neighbor machine learning model has been utilized to detect DDoS attacks in KDNs; however, it has resulted in lower classification performance compared to the J48 classifier [138]. In heterogeneous vehicular networks that use a KDN paradigm, cell selection for 5G millimeter wave communication considering the motion of nodes and characteristics of base stations has been realized by classification using a K-nearest neighbor model [139]. In KDN-based internet service provider networks, Transmission Control Protocol-Synchronization (TCP-SYN) and Internet Control Message Protocol (ICMP) flood attacks have been detected using K-nearest neighbor [140]. Detection of known and unknown saturation attacks by properly selecting a time window for OpenFlow traffic detection has been realized with the aid of a K-nearest neighbor classifier, where unknown saturation attacks have been detected by appropriately selecting a time window [141]. This research suggests that the time window of the traffic has a relationship with the model's classification performance. Even though packet classification has been achieved using a K-nearest neighbor classifier, its performance has been inferior to that of DNNs, which proves that it is less efficient at handling large datasets and the classification rules that occur in network scenarios [104]. To cope with the problem of overloading of the controller, a prediction mechanism for predicting harmful long-term loads of the controller in order to offload the controller in a harmful long-term scenario using a K-nearest neighbor classifier has been proposed; however, the prediction performance has been slightly inferior to SVMs [142]. K-nearest neighbor has been utilized to predict the Quality of Experience (QoE) in video data in real-time in KDNs [143]. Traffic classification as a knowledge generation step by collecting flow features such as protocol percentage, packet count, packet size, IP diversity ratio, etc. using a K-nearest neighbor has been realized in [144], in order to classify different IoT devices and detect different DDoS attacks such as TCP-SYN, UDP, and ICMP.

Unsupervised Learning Approaches for Generating Knowledge

K-Means Clustering

K-means is an unsupervised machine learning algorithm for grouping unlabeled data into a specified number (K) of clusters [145]. A multi-controller placement problem minimizing latency where an optimized K-means model is used for network partitioning in order to place the controllers has been studied in work [146]. Similar work uses an optimized version of the K-means algorithm for achieving multi-controller placement in KDN while minimizing the latency, which also guarantees high reliability and prevents the single point of failure problem [147]. K-means has been used for anomaly detection in smart grid KDN environments, which has resulted in high classification accuracy [148]. A DDoS attack detection mechanism under an unbalance in the traffic distribution using K-means clustering in a KDN has been studied in [149].

Isolation Forest

An isolation forest is a tree-based technique specially designed for anomaly detection that uses the average of the predictions by several decision trees to obtain the final anomaly score for the input data point [150]. An isolation forest classifier has been used for classifying network traffic's anomalies in a KDN–Network Function Virtualization (NFV) environment, which detects flooding attacks such as HTTP Flood, UDP Flood, Smurf Flood, and DDoS Flood [151]. An isolation forest has been used as an outlier detection scheme in combination with a data flow collection module and an information gain feature selection module to filter out the most relevant features to detect network security threats with high accuracy and low computational complexity [152]. A malware detection isolation forest using flow data of packets in combination with an automatically generated whitelist to eliminate benign packets in order to reduce the false positive rate of the isolation forest ML model has been studied in [153].

Semi-supervised Learning Approaches for Generating Knowledge

Variational autoencoders and generative adversarial networks stand out as generative models in semi-supervised learning.

Variational Autoencoder

A variational autoencoder (VAE) is a type of generative model that has an autoencoder with an encoder and decoder architecture that uses a probabilistic approach to encode input data into a latent space and then generate new data by decoding samples from the latent space while using variational inference to optimize the model during training [154]. A variational autoencoder for abnormal traffic detection that divides the traffic into 4 major categories and 38 specific attacks to be used as input to the decoder and has a higher classification accuracy and a higher generalization to detect new attacks is presented in [155]. In research [156], two approaches are studied to detect DoS and DDoS attacks. In the first approach, a variational autoencoder is trained on traffic traces of various types in order to learn the general features of the traffic. In the other approach, a variational autoencoder is used to learn abstract features from legitimate traffic in order to learn a representation of harmless traffic.

Generative Adversarial Networks (GANs)

Two neural networks serve as the generator and discriminator in a generative adversarial network. In a GAN, the generator is taught to produce new data that are similar to training data, and the discriminator is trained to distinguish between synthetic data and real training data [157]. A framework known as ByteSGAN, which consists of a generative adversarial network for encrypted traffic classification embedded in a KDN edge gateway and has better classification accuracy compared to CNN, has been discussed in [158]. While considering the network congestion in a heterogeneous knowledge-defined vehicular network, a generative adversarial network based method is used for augmenting the data dynamically used in creating network slices [159]. Generative adversarial networks have been used to synthesize realistic attacks using known attacks, such that attack variations have been trained to be detected in KDNs [160]. Considering the fact that DNNs tend to misclassify attacks under adversarial attacks, a generative adversarial network is used for

detecting DDoS under adversarial training to reduce errors in misclassifying adversarial attacks in a KDN [161].

Laplacian SVM

A Laplacian SVM is a semi-supervised machine learning approach that is a variant of the SVM that uses the Laplacian matrix to improve the accuracy of classification on complex and noisy datasets where labeled data are used to train the SVM while unlabeled data are used to construct the Laplacian matrix [162]. Laplacian SVM has been used for traffic flow categorization into QoS classes with the aid of deep packet inspection, where Laplacian SVM is specifically utilized to deal with traffic with unknown applications [163].

Self-Training

Self-training is a semi-supervised machine learning approach utilized in situations where there is a small amount of labeled data and a large amount of unlabeled data, such that the classifier is initially trained on the labeled data to predict labels for the unlabeled data, where the predicted labels are added to the labeled dataset, and the classifier retrained iteratively [164]. A self-learning-based traffic classification framework that uses dataset features such as flow meta-data information, size, and interval arrival times, where training using only 20% of the data as labeled data has produced satisfactory classification results [165].

Integration of Supervised with Supervised, Semi-Supervised, or Unsupervised Learning Approaches for Generating Knowledge

By combining a K-nearest neighbor (supervised) machine learning model with a variational autoencoder (semi-supervised) using a hybrid approach, both known and unknown DDoS saturation attacks have been detected in KDNs with a better performance than each of the machine learning model's individual performances [166]. An autoencoder and a one-class SVM are used for intrusion detection in the knowledge plane, while the P4 programming language along with ML have been used to implement real-time intrusion detection in the data plane [167]. A conditional variational autoencoder is used for data generalization in combination with a random forest classifier to automatically learn similarity among input features, extract discriminative features, and classify various types of attacks [168]. An intrusion detection model using a combination of a conditional variational autoencoder and a deep neural network where the decoder generates new attack samples to balance the training data, improving the detection rate of imbalanced attacks, has been effective in detecting minority attacks and unknown attacks [169]. To deal with both problems of an unbalanced and unlabeled data set in training for DDoS detection in KDNs, a combination of an autoencoder and a one-class SVM has achieved high accuracy with a small set of flow features [170]. Adversarial attacks have been generated and detected using a combination of generative adversarial networks and reinforcement learning in a wireless sensor network [171]. A flooding DDoS attack detection framework by combination of generative adversarial networks and deep learning, where the deep learning model has been trained incorporating adversarial attacks to improve the accuracy of classification [172]. A triple combination of a deep sparse autoencoder for dimension reduction, a generative adversarial network to solve the data imbalance problem, and, finally, ensemble learning using a supervised machine learning model has been used for the classification of anomalies in network traffic for KDNs [173]. A group of features in traffic flows, namely inter-packet arrival time, packet size, packet count, and flow tuple, have been used for flow classification using a combination of C4.5 decision trees and K-means clustering [174]. A management framework called ATLANTIC uses information theory to calculate deviations in the entropy of flow tables, in which a K-means classifier has been used for clustering traffic flows, followed by a SVM for classifying the flow of each cluster to detect and categorize traffic anomalies [175]. A modular DDoS attack detection system using K-means++ and fast K-nearest neighbor unsupervised and supervised machine learning combinations has resulted in high precision and stability for detection [176]. Ensemble learning is usually a combination of a machine learning approach with another machine learning approach of the same category (e.g., supervised with supervised) for

improving the accuracy and robustness of the predictions. An anomaly detection algorithm by combining X-means and isolation forest ML models using ensemble learning has shown better classification performance compared to the other unsupervised machine learning approaches' classification performance considered individually [177]. An ensemble learning approach has been proposed by combining K-nearest neighbor, Naive Bayes, support vector machines, and Self-Organizing Maps (SOMs) to detect anomalous behavior of data traffic in KDN, where the combination of SVM and SOM has resulted in high accuracy and detection rates with a low false alarm rate [178]. Ensemble learning using a random forest and gradient boosting has resulted in better performance than individual supervised machine learning approaches for Virtual Private Network (VPN) traffic classification [179].

Reinforcement Learning Approaches for Generating Knowledge

Reinforcement learning involves an agent in a given state taking actions to maximize future rewards obtained from the environment. Deep reinforcement learning combines ANN and reinforcement learning to achieve both function approximation and target optimization, mapping states and actions leading to rewards [180]. A Deep Reinforcement Learning (DRL) framework is utilized to train Graph Neural Networks (GNNs) using prioritized experience replay from the experiences learned by the controllers to predict the optimum routing path with the minimum average delay between source and destination nodes [181]. Similarly, another research highlights how closed loop control can be used for automatic routing in a KDN using deep reinforcement learning with the Deep Deterministic Policy Gradient (DDPG) to learn experiential knowledge, which also includes network monitoring to realize the interaction with the environment [44]. Routing optimization by adding a graph neural network to learn using deep reinforcement learning for message passing, which interacts with the network topology environment and extracts exploitable knowledge using the message passing process of information between the links with the goal of load balancing of network traffic, has been studied for KDNs in [182]. Ipv6 low-power wireless personal area networks have a routing protocol called the routing protocol for low-power lossy networks, which determines routes based on rank values and is thus vulnerable to rank attacks, which create non-optimized routes for packet forwarding. An alternative routing framework for the low-power wireless personal area network using the KDN paradigm, which uses reinforcement learning with Direct Acyclic Graphs (DAGs) for routing optimization at the controller having a QoS provisioning packet forwarding scheme preventing rank attacks, has been presented in [183]. A knowledge plane has been added to the SDN where reinforcement learning considering link-state information is utilized for making routing decisions, taking advantage of the global view and control in the KDN paradigm [184]. A reinforcement learning routing algorithm solving a traffic engineering problem in terms of throughput and delay using experience, where an agent learns a policy to suggest better routing paths, has been studied in [185]. To resist the dynamic change in flow control rules when the network is under attack, a deep reinforcement learning-based QoS-aware secure routing protocol has been utilized in a KDN that extracts knowledge from the history of traffic demands to dynamically optimize the routing policy [186]. Using real-time information of network state and flow statistics, a multi-path routing scheme for different flows by training using the Markov Decision Process (MDP) and Q-learning has been used in [187]. In the preceding scheme, remaining flows are redistributed according to the QoS priority to complete multi-path routing when there is no link that satisfies the bandwidth requirement. A global load-balanced routing scheme using reinforcement learning that makes a global policy for routing and load-balancing considering network resources for all flows coming from different sources has been studied in [188]. Multi-objective optimization and deep Q-learning are utilized in a multicast routing framework where multicast tree state, link bandwidth, link delay, and link packet loss rate are considered as the state-space and links are considered as the action space, where the knowledge is generated in the form of a multicast tree to be used by the controller for installing flow rules [189]. A traffic engineering framework uses a reinforcement learning-based scheme to learn a policy to select critical flows for a given

traffic matrix in order to reroute those critical flows using linear programming [190]. Using traffic distribution information collected by the controller, a deep reinforcement learning algorithm is used to dynamically adjust a set of links' weights for critical links as a knowledge generation step, where a weighted shortest path algorithm uses such knowledge to generate forwarding paths [191]. A general framework for security management using reinforcement learning, where a deep neural network-based Q-learning agent has been used to mitigate advanced persistent threats [192]. Multi-agent deep deterministic policy gradient-based deep reinforcement leaning has been used in KDNs for joint routing optimization and DDoS attack detection [193]. An anomaly detection and mitigation system uses reinforcement learning to learn policies for dealing with anomalies based on rewards for each action for each profile grouped based on collected network metrics [194]. A reinforcement learning framework has been used in a multimedia-based KDN environment to select the routing algorithm for QoS based traffic flows for QoS provisioning [195]. Deep reinforcement learning has been utilized to learn and build knowledge for 5G and beyond network slicing optimization where generated knowledge has been utilized for making optimal decisions [196]. For 5G cellular Radio Access Network (RAN), closed-loop power adjustment of radio transmitters has been realized in an automated and self-configuring manner by employing a deep reinforcement learning agent to develop intelligence with the objective of increasing equipment throughput [197]. Joint optimization of bandwidth allocation and position overlap of ultra-reliable low latency communication users in 5G networks has been achieved with the aid of deep deterministic policy gradient based deep reinforcement learning to generate knowledge by observing channel variations and traffic arrivals [198].

Now, let us analyze the task-wise classification of knowledge generation techniques discussed above. Table 2 summarizes the details of machine learning or heuristic model-based methods for different knowledge generation scenarios in KDNs.

ML models that are capable of generating knowledge are usually trained by vendors, who may keep the models proprietary. However, keeping them proprietary can cause redundant computations and an inefficient use of resources [199]. ML or heuristic models can be used to detect features from input information/data to create knowledge. However, this generated knowledge alone may not be sufficient to arrive at decisions regarding network functions. Thus, there is a requirement for a knowledge composition plane to further analyze and combine the different pieces of generated knowledge.

Table 2. Analysis of task-wise classification of knowledge generation in KDNs.

Task	Learning Class	Learning Technique	Special Purpose	Performance
Packet classification (PC)	Heuristic model	ABV algorithm	Parallelize ABV algorithm on GPU for PC [68]	99.9 times speedup, 65 times enhancement in throughput
		Lookup and BCAM	Packet classification using BCAM [69]	5 times fewer CAM bits than TCAM-based scheme
		Data structure RQ-RMI	Multi-field packet classification [70]	High compression factor, improvement in throughput
		DDBS algorithm	Multi-tuple packet classification [71]	Throughput: Over 10 Gbps on Cavium, 135 Gbps Xilinx
		Cross product, linear search	Packet classification [72]	Feasible, scalable, take less time and space
		TSS assisted algorithm	Packet classification [73]	Comparable update, high classifi. performance wrt. TSS
		Interval valued fuzzy logic	Classify video streams [74]	Reasonable accuracy, flexible, low computational cost
	Supervised ML	Deep neural network	Packet classification [104]	More than 90% classification accuracy
		Decision tree	CutSplit: DT combining cutting, splitting for PC [118]	10× memory reduction, 3× impro. of memory accesses

Table 2. *Cont.*

Task	Learning Class	Learning Technique	Special Purpose	Performance
Packet classification (PC)	Supervised ML	Decision tree	Bitcuts: Bit level cutting for PC [119]	2× throughput, 42–59% memory accesses wrt. others
		Decision tree	Hypercuts: Bit level cutting for PC [120]	2–10 less mem., 50–500% better search time wrt. HiCuts
		Support Vector machines	PC based on action/flow [129]	Less memory consumption, inferior accuracy than RF
		K-nearest neighbor	packet classification [104]	Accuracy inferior to DNN
Routing	Heuristic model	Extended Dijkstra algorithm	Find shortest path between source, destination [75]	Low end-to-end latency
		MO particle swarm optimization	Optimization based segment routing [76]	Reduce path consumption, better load balancing
		Greedy flow routing algorithm	Allocate path to flows using perform. thresholds [77]	Computationally complex, flexibly provides flow requirements
		Multi-path routing algorithm	Packet loss, time delay, bandwidth constrained routing[78]	35–70% improvement in quality-of-service
		QoS optimization routing algorithm	Routing satisfying video QoS requirements [79]	Low packet loss, good bandwidth utilization, high QoE
		Ant colony optimization algorithm	Dynamic routing [80]	Throughput, delay better than Dijkstra
		ILP maximizing energy ratio	Energy efficient routing [81]	Savings in 30% energy ratio, 14.7 W per switch, 38% link
	Supervised ML	Deep neural network	Find most reliable route in a SON [45]	90% accurate forecast in reliability prediction
		Deep neural network	To replace optimization models for routing [110]	Achieve quasi-optimal performance
		Deep neural network	3D two space division, forwarding for FANETs [111]	Better performance in packet delivery rate, energy-saving
		Deep neural network	Hybrid stable delay and distance based routing [112,200]	High packet delivery ratio, low latency and communication cost
		Deep neural network	Predicts a traffic matrix in real-time [113]	Similar perform. to heuristic routing, less execution time
	Reinforcement learning (RL)	Deep RL to train GNN	Prioritize experience replay to predict opt. path [181]	Good performance compared to Q learning, shortest path routing
		Deep RL with DDPG	Closed loop control for automatic routing [44]	Improved throughput, reduced packet delay
		Deep RL to train GNN	Routing with goal of load balancing [182]	Improved network performance
		Reinforcement learning	Aid link-state info. for making routing decisions[184]	Outperform Dijkstra by stretch, throughput, packet loss, delay
		Deep RL	Agent learns a routing policy by experience, reward [185]	Obtain higher rewards, transfer large files faster
		MDP and Q learning	Multi-path routing for different QoS flows [187]	Good jitter, packet loss rate wrt. ECMP, SP routing
		Reinforcement learning	Global load balanced routing scheme [188]	Outperform existing approaches wrt. delay, network utilization
		Deep Q-learning, optimization	Generate multicast tree for installing flow rules [189]	Better bandwidth, delay, packet loss rate
		Reinforcement learning	Critical flow rerouting [190]	Near-optimal performance by rerouting 10–21.3% of traffic
		Deep Q-learning, optimization	Critical links weight dynamically adjusted by RL [191]	Scalable,reduces transmission delay up to 39%
		Reinforcement learning	Select routing algorithm for QoS-based traffic flows [195]	Best trade off between QoS vs. QoE of a TC

Table 2. *Cont.*

Task	Learning Class	Learning Technique	Special Purpose	Performance
Security	Heuristic model	Optimization algorithm	Traffic sampling to inspect malicious network traffic [82]	Significantly outperforms equal rate all switch sampling
		Fuzzy logic-based algorithm	Security management, IDS [83]	More accurate results than algorithms used alone
		Attack detection algorithm	Kangaroo—Flow and packet based IDS [84]	Good attack detection rate, scalable
		Lion optimization	Detect DDoS attacks [85]	96% accuracy
		Fuzzy logic with InfoGain FS	Anomaly-based intrusion detection [86]	91.1%—true-positive rate, 0.006%—false-positive rate
		Evolving fuzzy system	Network anomaly detection [87]	81%—binary classification, 80% multiclass classification accuracy
	Supervised ML	Deep neural network	Network intrusion and anomaly detection [101]	Better throughput, latency, resource utilization
		Convolutional neural network	To detect DDoS attack [103]	High detection rate—DNS amplification, NTP, TCP-SYN
		GRU–LSTM-based DNN	Flow-based anomaly detection [107]	Accuracy—87.9%, precision—99.8%, recall—99.4%, F1-score—99.2%
		Decision tree	To detect DDoS attack [117]	Better performance compared to SVM and Naïve Bayes
		Decision tree	Vulnerability detection using traffic classification [121].	Good accuracy, training time, prediction speed
		Support Vector Machine	Flooding-based multiclass DDoS detection [125]	97% accuracy with fastest training, testing time
		Support Vector Machine	Hierarchical flow, packet-based anomaly detect [126].	Good detection rates, minimal extra overhead
		Random forest	Detect DDoS attack [133]	High false positive rate, low accuracy wrt. SVM
		Random forest	Intrusion detection system [134]	Superior performance to NN and Naive Bayes
		K-nearest neighbor	Detect DDoS attacks [138]	Lower classification performance wrt. to J48 classifier
		K-nearest neighbor	Detect TCP-SYN, ICMP flood attacks [140]	Highest F1-score wrt. NN, DT. Mitigate 98% attacks
		K-nearest neighbor	Detect known, unknown saturation attacks [141]	For 1 min TW: precision: 96%, recall: 95%, F1 score: 95%
		K-nearest neighbor	Classify devices and detect DDoS attacks [144]	Accuracy 97%: device classification, 98%: DDoS detection, latency: 1.18 ms
	Unsupervised ML	K means clustering	Anomaly detection in smart grid KDN [148]	High accuracy (96.9%)
		K means clustering	Detect DDoS attacks [149]	Recall < 90%, CPU utilization—12%, Maximum packet loss rate—7.5%
		Isolation forest	Classify traffic anomalies in a NFV environment [151].	TPR—0.8708, FPR—0.0258
		Isolation forest	Detect network security threats [152]	High accuracy, detection rate, low computational complexity
		Isolation forest	Malware detection supported by a whitelist [153]	TPR—0.9998, FPR—0.0325
	Semi-supervised ML	Variational autoencoder	Abnormal traffic detection [155]	Accuracy rate of 87.27%
		Variational autoencoder	Detect DoS and DDoS attacks [156]	Benign traffic—97%, Malicious traffic—93% accurate
		Generative adversarial networks	Synthesize and train to detect attacks [160]	0.51 detection rate after 95 epochs
		Generative adversarial networks	Adversarial training to detect DDoS attacks [161]	Accuracy—99.8%, Precision—99.8%, Recall—99.9%, F1score—99.9%

Table 2. *Cont.*

Task	Learning Class	Learning Technique	Special Purpose	Performance
Security	Integrated ML	KNN, Variational autoencoder	Detect known, unknown DDoS attacks [166]	Precision—0.85, Recall—0.97, F1-score—0.91
		Autoencoder, 1-class SVM	Intrusion detection [167]	Detection accuracy of 97%
		Conditional VAE, random forest	Attacks classification with generalized training [168]	Precision above 99%, better performance than SVM + RF
		Conditional variational AE, DNN	IDS including minority and unknown attacks [169]	Accuracy of 89.08%, DR of 95.68%
		AE, 1-class SVM	DDoS detection under imbalanced, unlabeled data [170]	Average accuracy of 99.35%
		GAN, Reinforcement learning	Detection of attacks including adversar. attacks [171]	Average accuracy of 85.40%
		GAN, DNN	Detecting DDoS attacks with adversarial training [172]	Adversarial training improves detection performance
		AE, CGAN, ensemble learning	Intrusion detection [173]	Fast classification with high detection accuracy
		K means, SVM	Detect and categorize traffic anomalies [175]	Accuracy of 88.7%, precision of 82.3%
		K means++, fast KNN	Modular DDoS attack detection [176]	High efficiency, precision, and stability
		Isolation forest, X-means	Anomaly detection [177]	Average AUC 8.1%, ADR 19.5% better than iForest, others
		KNN, NB, SVM, SOM	Anomaly detection [178]	SVM-SOM: 98.1% accuracy, 97.1% DR, FPR—2.7%
	Reinforcement learning	RL with DAG	Routing optimization preventing rank attacks [183]	PDR—85%, delay—1.5–2.9 s, prevent ranking attacks satisfactorily
		Deep reinforcement learning	QoS-aware secure routing [186]	QoS performance gains when network under attack
		DDPG based DRL	Joint routing optimization and DDoS detection [193]	MA-better packet delay, jitter, packet loss, detection rate
		Neural fitted Q-learning	ATMoS: Autonomous threat mitigation [192]	The model converges, can detect and mitigate actors of APTs
		Reinforcement learning	Anomaly detection [194]	Detect attacks, state representation has a high storage cost
Controller Placement (CP)	Heuristic model	Evolutionary algorithm	Multi-objective controller placement [88]	Needs less memory, computation time, higher coverage
		Heuristic optimization algorithm	Reliable fault-tolerant controller placement [89]	Computation time < 2 min, controller number, location vary on topology
		Binary ILP, heuristic algorithm	Energy aware controller placement [90]	BILP is computationally complex, heuristic algorithmic solution is near optimal
		Particle swarm optimization, firefly algorithm	CP considering latency, multipath connectivity [91]	Minimizes average delay during single link failure
	Unsupervised ML	Optimized K-means	Partition algorithm for controller placement [146]	Maximum latency 2.437× less than standard K-means
		Optimized K-means	Controller placement minimizing latency [147]	Latency less than standard K-means, high availability

Table 2. *Cont.*

Task	Learning Class	Learning Technique	Special Purpose	Performance
Traffic Classification (TC)	Supervised ML	Deep neural network	TC for energy efficiency optimization [106]	47.71% energy consumption reduction, good load balancing
		Decision tree	Online TC using flow-level features [122]	97.92% accuracy, 7500+ Million Classes
		DT in Hadoop platform	Big data traffic classification [123]	Faster and accurate than C4.5 decision tree
		Support Vector Machines	Application-/service-wise traffic classification [128]	Classify 28 applications with 85.98% accuracy
		Support Vector Machines	Traffic classification using traffic statistics [130]	Accuracy over 95%, 4% F1-score for attacks
		Random forest	Internet protocol traffic classification [135]	Classification accuracy varied 73–96% for different apps
		Random forest	Classify application QoS from switch statistics [136]	Prediction error less than 10%
	Semi-supervised ML	Generative adversarial networks	ByteSGAN-Encrypted traffic classification [158]	Accuracy more than 90% and better than CNN
		Self-learning	Application-wise traffic classification [165]	20% LD: Accuracy—60%. High recall, similar precision wrt. supervised learning
	Integrated ML	Ensemble learning with RF and GB	VPN (secure) network traffic classification [179]	93.80% accuracy, Precision—91.80%, recall—96.20%, F1-score—94%
Fault/Failure diagnosis	Supervised ML	Deep neural network	Detect link failures [108]	85% accuracy in identification of failures
		LSTM-based DNN	Detect link handover failures in advance [109]	99% TPR, predict a link failure before 1–2 s
		K nearest neighbor	Harmful long-term load prediction [142]	Accuracy—96.2%, Precision—91.2%, F1-score—94.6%
QoS/QoE prediction	Supervised ML	GRU-DNN	Temporal-aware service QoS prediction [114]	Superior accuracy wrt. other approaches
		DNN	Multiple attributes QoS prediction [115]	Low mean absolute error wrt. other approaches
		K nearest neighbor	Predict QoE in video data [143]	k = 1, Pearson Correlation Coefficient—0.75, long testing time
	Semi-supervised ML	Laplacian SVM	Classifies network traffic based on QoS requirements [163].	Accuracy exceeds 90% and better than K-means
WL handover	Supervised ML	K nearest neighbor	Adaptive cell selection for mm Wave HetNets [139]	Better handovers—45.83%, throughput—17.2%, EE—16.7%
Network slicing	Semi-super. ML	Conditional Generative adversarial networks	CGAN augment data used in creating network slices [159]	Accuracy range in 0.40–0.65

3.4. Knowledge Composition Plane

In the knowledge composition plane, synthesized knowledge from machine learning models/heuristic methods is further analyzed or collected and combined to produce new knowledge. This knowledge-composing element is called the ontology editor, which combines universal knowledge (already existing knowledge) with multiply generated knowledge. An ontology editor is a software application designed to create and maintain ontologies that can be used to define the structure of the ontology by identifying classes, properties, relationships, and hierarchies within the domains using an ontology language such as OWL [201]. For example, consider the cruise control application in autonomous driving. Assume that there is a ML model that takes a camera image and the speed of the vehicle as input and generates a forward collision warning as output (knowledge). Consequently, the ML model will provide a warning signal when the car is moving and there is an item in front of the camera (a person or another vehicle). The vehicle's acceleration may

not be able to be controlled by this warning signal alone. This knowledge may have to be combined with other knowledge such as collision warnings received from other sensors such as proximity sensors, warnings received from RSUs, output from another knowledge model (a trained ML algorithm), etc. in order to decide the cruise control value. The composed knowledge by the ontology editor can be represented and modeled in numerous ways, which can be selected based on the type of knowledge and design requirements.

3.4.1. Composed Knowledge Representation

The composed knowledge can be represented mainly in four different forms, which are described in the following sub-sections. Figure 4 depicts different formats in which knowledge can be represented.

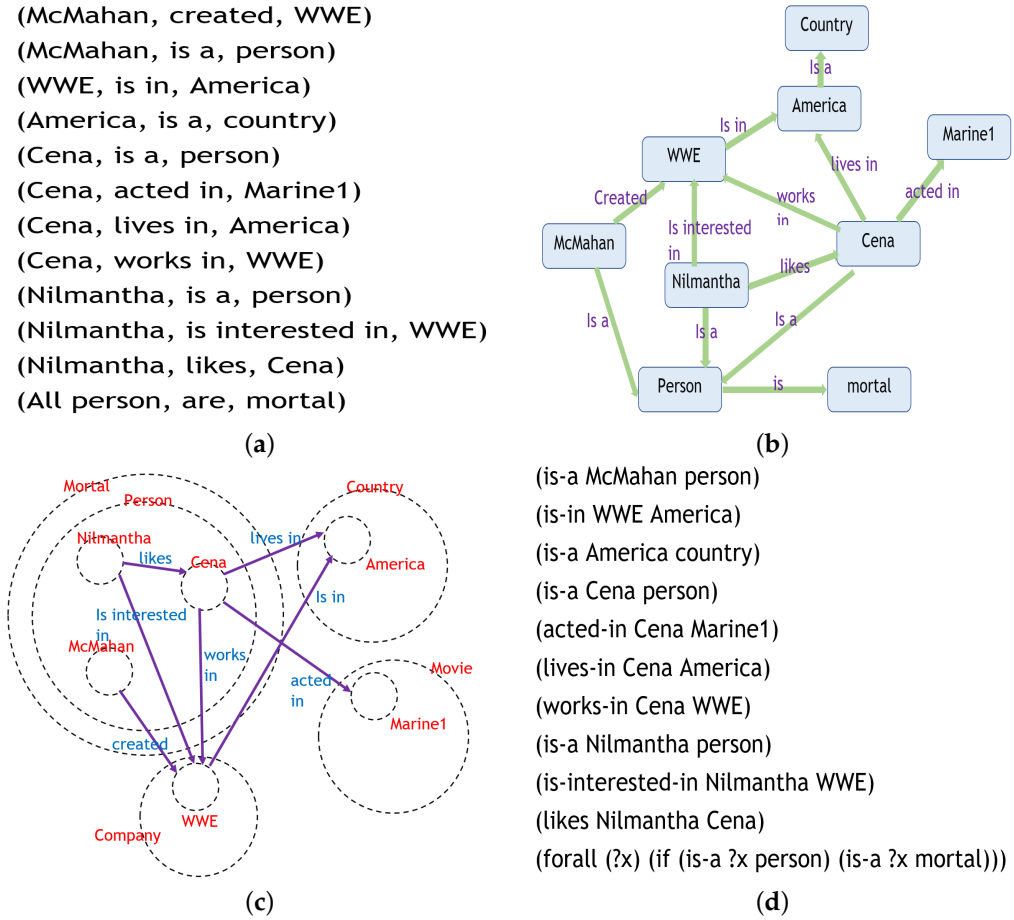

Figure 4. A sample of knowledge represented in different knowledge models. (**a**) Knowledge represented in Resource Description Framework (RDF) format. (**b**) Knowledge represented as a Knowledge graph. (**c**) Knowledge represented as an ontology. (**d**) Knowledge represented in Knowledge Interchange Format (KIF).

Resource Description Framework (RDF)

RDF is the format in which generated knowledge is basically represented. This basic knowledge representation can also be used to represent the composed knowledge. However, this format is not recommended, as it is difficult to make decisions directly from the knowledge that appears in this format, as knowledge appears in basic triplet form and is not classified into classes, as evident from the example in Figure 4a. Therefore, it is recommended to represent the composed knowledge in any other format, such as a knowledge graph, ontology, or knowledge interchange format, which can be queried easily or used readily for making decisions.

Knowledge Graphs

A knowledge graph represents the composed knowledge in a structured and machine-readable format. It consists of entities/nodes (representing concepts) and edges (representing relationships between the concepts), as shown in Figure 4b. Knowledge graphs are used to represent complex relationships between entities and can be used for semantic reasoning. One of the key benefits of using a knowledge graph in KDN is that it allows for flexible reasoning about the network. The graph structure enables sophisticated queries that can traverse multiple levels of abstraction. However, knowledge graphs can be limited in their ability to represent complex relationships and hierarchies [202]. Knowledge Graph Schema (KGS), collections of rules and standards that describe the structure of a knowledge graph, may be used to generate knowledge graphs. They offer a standardized method for representing entities, connections, and characteristics in a knowledge graph, assisting in the maintenance of consistent, accurate, and simple-to-understand data. A KGS can include a variety of components, such as classes, properties, and restrictions. An example of a KGS is the RDF Schema (RDFS) language, which provides a vocabulary for describing classes, properties, and their relationships [203].

Ontologies

Ontologies define a set of concepts and their relationships that describe a particular domain. The Web Ontology Language (OWL) or Ontology Inference Layer (OIL) can be used to model and describe ontologies. OWL builds on knowledge that appears in the form of RDF to create a complex knowledge structure by clustering knowledge into domains (classes), as shown in Figure 4c. Thus, generated knowledge by machine learning models that appears in RDF format can be readily converted into composed knowledge as an ontology using OWL [204]. OWL has basic elements such as classes, properties, individuals (class instances), restrictions, and annotations. On the other hand, OIL extends RDF and provides a lightweight ontology language that consists of basic elements such as classes, properties, individuals, restrictions, and inference rules. However, OIL is less expressive than OWL, which means that OWL is capable of representing a wider range of concepts and relationships due to its richer syntax [205]. OWL has a richer syntax, as it allows the definition of property characteristics, property restrictions, datatype properties, etc. The use of ontologies in KDN enables the network to reason about the current state of the network and to make decisions based on that knowledge. Ontologies can be used to capture domain-specific knowledge, which can then be shared and reused across different network applications. However, ontology representation can be limited in its ability to represent uncertainty and ambiguity [206].

Knowledge Interchange Format (KIF)

KIF is a logic-based representation language that uses a syntax similar to mathematical notation, allowing for the specification of complex relationships between objects and concepts to represent knowledge. A sample knowledge representation using KIF is shown in Figure 4d. KIF is a formal language for representing and exchanging knowledge that is designed to be unambiguous and independent of any particular knowledge representation system, making it more flexible than other approaches. KIF is primarily useful for the representation and exchange of knowledge between different AI systems, as this knowledge representation is more machine-understandable [207]. However, a high level of human expertise is required to understand knowledge in KIF format due to its logical representation.

3.4.2. Rule Generation

Note that, as evident from Figure 3, the knowledge composition plane responds to users'/ applications' demands by comparing users'/applications' demands with the composed knowledge to produce new rules to be used by other planes (application, control, and management). The user's intent can appear as the user's objective and operation, which can be expressed using a high-level declarative language such as Structured Query Language (SQL) and conveyed to the knowledge plane using an API [25]. For example, if the

composed knowledge appears as the network congestion values of links in the network, the network administrator may set a network congestion threshold value to be compared with the composed knowledge in the form of a network congestion value in order to produce a rule that instructs nodes to increase their energy utilization if the network congestion is greater than the threshold. In this example, note that the users' objective is to control network congestion, while the users' operation is to control energy utilization. The operation will be triggered in order to achieve the desired objective. However, applications'/users' objectives and operations could be high-level and complicated, unlike in the specified example given above. Therefore, these need to be parsed and converted to multiple decomposed elements to be processed by the rule generator, as shown in Figure 3. Then, composed knowledge and the users' intent will be orchestrated to produce rules by the rule generator. Note that the outputs of the knowledge composition plane are rules that can be directly used by other planes without further processing and composed knowledge that can be stored in the knowledge distribution plane and provided to application, management, and control planes as required.

Typically, a rule generator can be implemented as a heuristic model with the aid of a programming language such as Java, Python, Lisp, Prolog, etc. However, a machine learning model can also serve the purpose of a rule generator by being appropriately trained to generate rules. However, the rules generated by a ML model are required to be translated into a rule language using natural language processing [208].

There should be a language for the rules generated by the rule generator in order for the rules to be understandable by other planes, such as the control plane. The generated rules can be represented in a dedicated rule language such as Rule Markup Language (RuleML) or Semantic Web Rule Language (SWRL). RuleML is a markup language used for representing rules in a machine-readable format. It allows users to express rules in a standardized way, making it easier to exchange rules between different rule engines and systems. RuleML also provides a set of tools and APIs for working with rule-based systems [209]. Semantic Web Rule Language (SWRL) is a rule language that is used to express rules and constraints and is based on a combination of OWL and RuleML. SWRL allows for the creation of rules that can be expressed in a natural language-like syntax and can be used to represent complex logical relationships between entities. One advantage of SWRL over other rule languages is that it has been designed to be compatible with RDF and OWL, so that generated rules can be utilized in combination with knowledge represented in those formats [210]. Another promising candidate for representing rules is the Rule Interchange Format (RIF). RIF is a language that was developed by the World Wide Web Consortium (W3C) to create a common rule language that can be used by different rule systems to exchange rules. RIF can be used to exchange rules between the control, application, and management planes and the knowledge plane in a knowledge-defined network [211].

3.4.3. Examples Using Composed Knowledge in the Existing Literature

In [212], a network-wide packet classification algorithm that uses a multi-valued decision diagram is used to classify packets, and the classification knowledge is used to generate rules for automatic fault localization. Similarly, another packet classification algorithm known as split bit vector packet classification functions by splitting and then classifying in parallel, and then that classification knowledge is orchestrated with the objective of minimizing latency in order to produce OpenFlow flow rules [213]. Generating rules for QoS management using the knowledge generated as traffic classification can be considered as a knowledge composition step. In [214], the network encrypted traffic in a smart home network is classified using three deep learning schemes: multilayer perceptron, stacked autoencoder, and a CNN where the knowledge on traffic classification is compared with application layer requirements in order to generate rules related to network management. Another similar work classifies traffic using deep learning, which is used to generate rules for fine-grained network management and resource utilization [215].

Machine learning or model-based methods can be used for intrusion detection in KDNs, as discussed in Sections 3.3.1 and 3.3.2. However, after detection, prevention of further damage or threat-suppressing actions belongs to the knowledge composition step. An intrusion prevention system that detects DDoS, worm spreading, and scanning on a global level in the network using a parallel neural network, and then the knowledge on the detection class is used to generate rules for taking necessary actions to suppress the attacks [216]. An intrusion prevention system using fuzzy logic that can decide the length of the blocks based on knowledge of the frequency and type of the attack, that will generate rules to block the attacker host for a specific period of time has been studied in [217]. Another similar work detects DDoS attacks using genetic algorithms and uses the knowledge of the attack class, composing it with other knowledge on the frequency and nature of the attack, to generate rules to block the attacking host for the duration of the attack [218]. An Address Resolution Protocol (ARP) spoofing attack is where adversaries send fabricated ARP messages linking the attacker's Media Access Control (MAC) address to a genuine device's Internet Protocol (IP) address. An intrusion detection and prevention system that detects ARP spoofing attacks by detecting malicious network traffic, where rules are generated to drop packets when the attack is detected [219].

In terms of routing in computer networks, the forward path selection can be considered as a knowledge generation step, while the flow-rule placement is a knowledge composition step that uses the knowledge generated in the form of forwarding paths to generate flow rules. In [220], a framework called FlowStat uses per-flow statistics to produce knowledge on optimal paths using max-flow min-cost optimization and then uses integer linear programming as the rule generator to decide forwarding rules for the forwarding paths. A routing framework that classifies the data flow into two classes (mice and elephant) based on their size and then, based on the knowledge of the classified flow type, uses a machine learning technique called association rules to generate forwarding rules for each flow class [221]. Ternary Content Addressable Memory (TCAM) modules have a limited capacity and are used to store flow rules in KDNs. In [222], first, knowledge is generated in the form of routes without considering TCAM constraints; then, that knowledge is used in a mixed-integer linear programming model to compute flow rules considering the limited capacity of TCAM. A security-oriented routing mechanism called RouteGuardian has been utilized in KDN, which has a network security virtualization framework to detect abnormal traffic and compose that knowledge with the latest network status to generate rules to isolate malicious nodes and reconfigure routes [223]. In another routing framework for KDN, a graph neural network is used for predicting link delay, where the knowledge on predicted delay is composed with adaptive flow splitting using the rule timeout mechanism in the OpenFlow protocol to generate forwarding rules [224]. A QoS-aware flow rule aggregation scheme called Q-Flag for knowledge-defined IoT networks first generates knowledge by selecting paths that minimize flow table utilization using a heuristic model-based approach, and then composes the generated knowledge considering both the flow rule capacity of the switches and the QoS requirements of applications to produce aggregated flow rules [225].

3.5. Knowledge Distribution and Management Plane

Knowledge management in the context of knowledge-defined networking refers to the process of capturing, storing, sharing, and utilizing knowledge and rules within a networked environment. Knowledge management aims to improve the performance of the networked environment by ensuring that the required knowledge and rules are provided to the necessary parties. It involves knowledge and rule sharing, which involves establishing processes and systems for capturing and disseminating knowledge and rules across the network. This plane basically consists of a Knowledge Base (KB) that necessarily consists of generated knowledge by knowledge generation models, composed knowledge by ontology editors, generated rules by rule generators, collected data from the network, and control messages, as evident from Figure 3. Data models such as Yet Another Next Generation

(YANG) or Common Information Model (CIM) can be used to store and disseminate network data, which are discussed in detail in Section 4.5.3. Raw data are received to this plane from the control and management planes, while the received data can be stored in the knowledge base and provided to the knowledge generation plane when required. Knowledge and rules can be shared from this knowledge base to the application plane, the control plane, the management plane, and the knowledge composition sub-plane (generated knowledge). Thus, it is required to have standards for knowledge and rules storing, sharing, and utilization in order for multiple KDN systems to be interoperable.

As the network has limited networking capabilities and resources, there should be transmission rules for transmission of knowledge related traffic (generated knowledge, composed knowledge, or rules generated from composed knowledge). One of the purposes of the knowledge distribution plane is to derive transmission rules for knowledge-related traffic to be exchanged within the knowledge plane and between the knowledge plane and the application, management, and control planes. For instance, rules generated by the knowledge composition plane are relatively smaller in size compared to descriptive knowledge produced from the knowledge generation plane. Therefore, transmission of generated rules from knowledge requires higher reliability, high priority, and high latency, whereas relatively larger-volume descriptive knowledge transmission may need low reliability and low priority, and may tolerate some higher latency. In other words, the data flows for each rule and descriptive knowledge have their own QoS parameters. QoS is the ability of a network to provide the required services for selected network traffic. The descriptive knowledge may be optionally distributed to end devices at the discretion of the controller. For example, in a vehicular network, if knowledge is generated related to the status of vehicular traffic congestion, that knowledge is required to be sent to all the nodes in the network controlled by the controller.

3.5.1. Review on Existing Work on Flow Scheduling and Prioritization to Apply Knowledge and Rule Dissemination

The OpenFlow protocol has different tactics to control QoS parameters in data flows. It has an enqueuing feature to maintain different queues for different flows, matching and tagging of Virtual Local Area Network (VLAN) and Multi Protocol Label Switching (MPLS) labels with traffic classes, rate limiting functionality by means of meter tables, and a flow monitoring framework [226]. However, OpenFlow does not provide support for queue configuration, as it is currently handled by OF-CONFIG, which has been standardized by the Open Network Foundation (ONF). There are a few open-source SDN controller projects with QoS support, such as OpenDaylight, Open Network Operating System (ONOS), Floodlight (FL), etc. In order to disseminate descriptive knowledge and generated rules having different QoS parameters, QoS-motivated routing with per-flow routing can be employed [79]. Dynamic routing of flows is achievable due to the decoupling of control and forwarding functions in a KDN architecture. QOGMP is a KDN control architecture that has a distributed control architecture, where there is a broker on the top level for traffic scheduling regarding flow coordination over multiple domains based on QoS requirements, which can be used to disseminate knowledge to end devices [227]. Some have attempted to give a high preference to high-priority traffic by allocating more resources while at the same time considering the end-to-end latency requirements by using weighting parameters, which is a joint resource allocation scheme for multiple traffic classes [228]. QoS APIs have been utilized as an extension to OpenFlow to control configuration and management of QoS parameters where aggregated bandwidth usage is accomplished by a rate limiter and flows are mapped to priority queues using queue mapping to tackle bandwidth and delay allocation [229]. Such QoS APIs can be utilized to disseminate knowledge and rules considering QoS parameters. In a multi-operator KDN environment where KDN controllers belong to different service providers, the KDN controllers directly communicate with each other to advertise a set QoS of their network to others where network service providers buy the best path from the other network for a price, which is an open exchange framework,

having collaborations between the KDN controllers where rules and knowledge can be effectively exchanged between the networks considering QoS requirements [230]. Packet marking such as collaborative borrowing, queue management such as weighted random early detection, queue scheduling such as weighted round robin, and priority queuing, can be used for knowledge and rule dissemination, adhering to QoS management in KDN environments [231]. Resource allocation using multi-objective optimization can be used to allocate resources for different network flows, such as rules and knowledge, in a heterogeneous network environment [232].

3.5.2. Knowledge and Rule Exchange and Management Protocols/Languages

The above-reviewed work can be used to effectively prioritize knowledge and rule flows in order to exchange knowledge and rules between planes in a KDN. However, in order for such flow prioritization and scheduling to be effective, knowledge and rule exchange, management, and retrieval protocols can be utilized to make knowledge and rule exchanging, retrieval, and management to be efficient, interoperable, scalable, and reliable.

As discussed in Section 3.4 on knowledge composition, RuleML, SWRL, and RIF can be used for all representing, storing, and exchanging rules. In the knowledge distribution and management plane, generated rules are required to be stored in a rule repository, which can be implemented using any of the above languages. Note that rules in the rule repository can be managed using insert, update, and delete operations of the rules using the same rule languages. Furthermore, other planes need to have a rule engine to evaluate the rules/knowledge transmitted from the knowledge plane and execute the actions. A rule engine's purpose is to execute rules or make inferences from knowledge and make decisions based on the rule/knowledge evaluation. Forward-chaining in a rule engine refers to using a set of rules to arrive at a conclusion, while backward-chaining refers to a goal-driven reasoning approach that determines the rules to be applied to reach a predetermined conclusion/goal. Given below are several existing rule/knowledge evaluation techniques/rule engines:

- RETE is an algorithm used for rule evaluation and pattern matching that builds a network of nodes that represent the conditions and actions of rules, where nodes are connected in a way to allow efficient evaluation of the rules [233];
- Drools is a Java-based open-sourced rule engine that supports advanced features such as query capabilities, rule chaining (forward and backward), and rule templating, which uses RETE as the algorithm for pattern matching [234];
- VLog is a rule-based reasoner with a platform-independent Java API that supports reasoning using existential rules or knowledge represented in RDF or OWL formats [235];
- Bossam is a rule engine that can reason from OWL ontologies and also provides support for forward and backward chaining [236];
- The C Language Integrated Production System (CLIPS) is a rule-based programming language. The CLIPS rule engine can handle large rule sets and can evaluate rules using a forward chaining algorithm to derive conclusions [237];
- Jess is a rule engine for the Java platform that infers conclusions/decisions from rules, similar to other rule engines [238].

High-level users, such as network administrators/applications, can retrieve knowledge or ask questions from a knowledge model using Knowledge Query Language (KQL), which allows applications/users to receive knowledge more easily and effectively without having to understand the details of the underlying knowledge implementation. Furthermore, some protocols allow knowledge modification (insert, delete, and update operations), allowing knowledge to be effectively managed in the knowledge exchange and management plane. Given below are the main languages/ protocols that can be used to query and/or modify knowledge in KDNs:

- SPARQL Protocol and RDF Query Language (SPARQL) is a query language for RDF to query knowledge models that appear in the form of RDF format. As generated knowledge can be modeled using RDF and ontologies derived from such knowledge

are based on OWL, which is based on RDF, SPARQL can be used to query knowledge contained in those forms. SPARQL further has an update language extension, which includes commands such as insert, delete, and update for modifying RDF-based knowledge [239];

- SQWRL is an extension of SWRL (Semantic Web Rule Language) that adds support for querying OWL ontologies. SQWRL allows users to query OWL ontologies using a combination of SWRL rules and OWL axioms. SQWRL also supports complex queries that combine multiple conditions using logical operators such as AND and OR. However, SQWRL is designed specifically for querying OWL ontologies and does not provide any opportunity to modify ontologies [240];
- Knowledge Graph Query Language (KGQL) is a query language specifically designed for querying knowledge graphs that allows users to write complex queries over knowledge graphs using a simple and instinctive syntax. KGQL supports pattern matching, filtering, sorting, aggregation, path traversal, etc. KGQL also does not support knowledge modification; thus, it is a read-only query language [241];
- Knowledge Query and Manipulation Language (KQML) is a message-passing protocol that allows agents to exchange knowledge and perform tasks in a distributed AI system, regardless of their underlying implementation. In KQML, a simple text-based syntax is used to represent different message types such as queries, assertions, and requests for action that can be exchanged using KQML. KQML is supposed to exchange knowledge and does not explicitly support knowledge modification [242];
- GraphQL is another language that can be used to not only retrieve knowledge from knowledge graphs, but also modify knowledge using insert, update, and delete operations. Similar to other query languages used to retrieve knowledge, GraphQL also allows the agents/developer to specify the knowledge in the form of a knowledge graph that needs to be retrieved, allowing fast and efficient data retrieval [243].

The above-given protocols/languages can be effectively utilized along with a flow prioritization and scheduling approach to exchange or retrieve knowledge and rules. Furthermore, knowledge can be managed efficiently using protocols that allow knowledge modification. For instance, old knowledge can be deleted to utilize limited storage in cases where such knowledge is not required for current decision-making. Furthermore, existing knowledge can be modified based on new policies, etc., with the aid of protocols such as SPARQL, GraphQL, etc.

Table 3 summarizes existing languages, models, and protocols for data, knowledge, and rules representation, modification, evaluation, and exchange.

Table 3. Summary of data, knowledge, and rule representation, modification, and exchanging languages, models, and protocols.

Category	Language/Protocol/Model	Purpose	Developer
Knowledge representation	Resource description framework (RDF) [64]	Represent knowledge as (subject, predicate, object) triplets	W3C
	RDF schema (RDFS) [203]	Represent knowledge using a knowledge graph having entities and relationships	W3C
	Web Ontology Language (OWL) [204]	Represent ontology using classes, properties, individuals, restrictions, annotations	W3C
	Ontology Inference Layer (OIL) [205]	Represent ontology using classes, properties, individuals, restrictions, inference rules	Dieter Fensel et al.
	Knowledge Interchange Format (KIF) [207]	Represent knowledge using complex logical relationship between objects and concepts	Genesereth et al.

Table 3. *Cont.*

Category	Language/Protocol/Model	Purpose	Developer
Rule representation, storage and exchange	RuleML [209]	Markup language to represent rules	H. Boley et al.
	Semantic Web Rule Language (SWRL) [210]	Express rules and constraints	W3C, RIF
	Rule Interchange Format (RIF) [211]	Rule language to represent and exchange rules between systems	W3C, RIF
Query and modify knowledge	SPARQL [239]	Query and modify knowledge which appears in RDF	W3C
	GraphQL [243]	Query and modify knowledge in knowledge graphs	Huahai He et al.
Query knowledge	SQWRL [240]	Query OWL ontologies	W3C
	Knowledge Graph Query Language (KGQL) [241]	Query knowledge graphs	P. Liu et al.
	Knowledge Query and Manipulation Language (KQML) [242]	Exchange messages such as queries, assertions, and requests	T. Finin et al.
Data representation	YANG [244]	Describe state data of network devices and their configurations	Bjorklund
	Common Information Model (CIM) [245]	Representing information about network infrastructure and services of devices	Uslar M. et al.
Rules/knowledge evaluation	RETE [233]	Rule evaluation and pattern matching	A. Gupta et al.
	Drools [234]	Rule pattern matching supporting querying, rule chaining	M. Proctor
	VLog [235]	Reasoning for rules or knowledge represented in RDF or OWL	D. Carral et al.
	Bossam [236]	Reason from OWL ontology, support chaining	M. Jang et al.
	CLIPS [237]	Evaluate rules using a forward chaining algorithm	F. M. Lopez
	Jess [238]	Rule engine for the Java platform	F.H. Ernest

4. Management Plane

4.1. Introduction to Management Plane

There is a requirement for efficient and effective management of the network resources in order for the control plane functionality to be effective. The main purpose of the management plane, which operates in parallel with the KDN controller in the KDN architecture, is to collect data and information from the devices in the network, discover and configure QoS options in the network, monitor the network devices, configure the network, prevent undesired traffic interruptions, etc. [58]. Traditional network management has been mostly performed by humans, which can be erroneous due to human errors. In such networks, the network operators must implement policies and complex tasks with a highly constrained set of configuration commands in a command-line interface environment. As the network state changes continuously, reconfiguring manually is time-consuming and less efficient. Note that the management plane has been identified as the "Measurement" plane by some researchers, which serves the same purposes as the management plane described above [57]. Therefore, note that measurement or management plane refers to the same plane. For the rest of this paper, we use the term management plane in order to maintain consistency in terminology, which refers to the same plane referred to in the existing literature as the measurement plane.

4.2. Architecture of the Management Plane

Figure 5 shows the expanded architecture of the management plane with detailed flows between each functional block in the management plane and other planes [57,58].

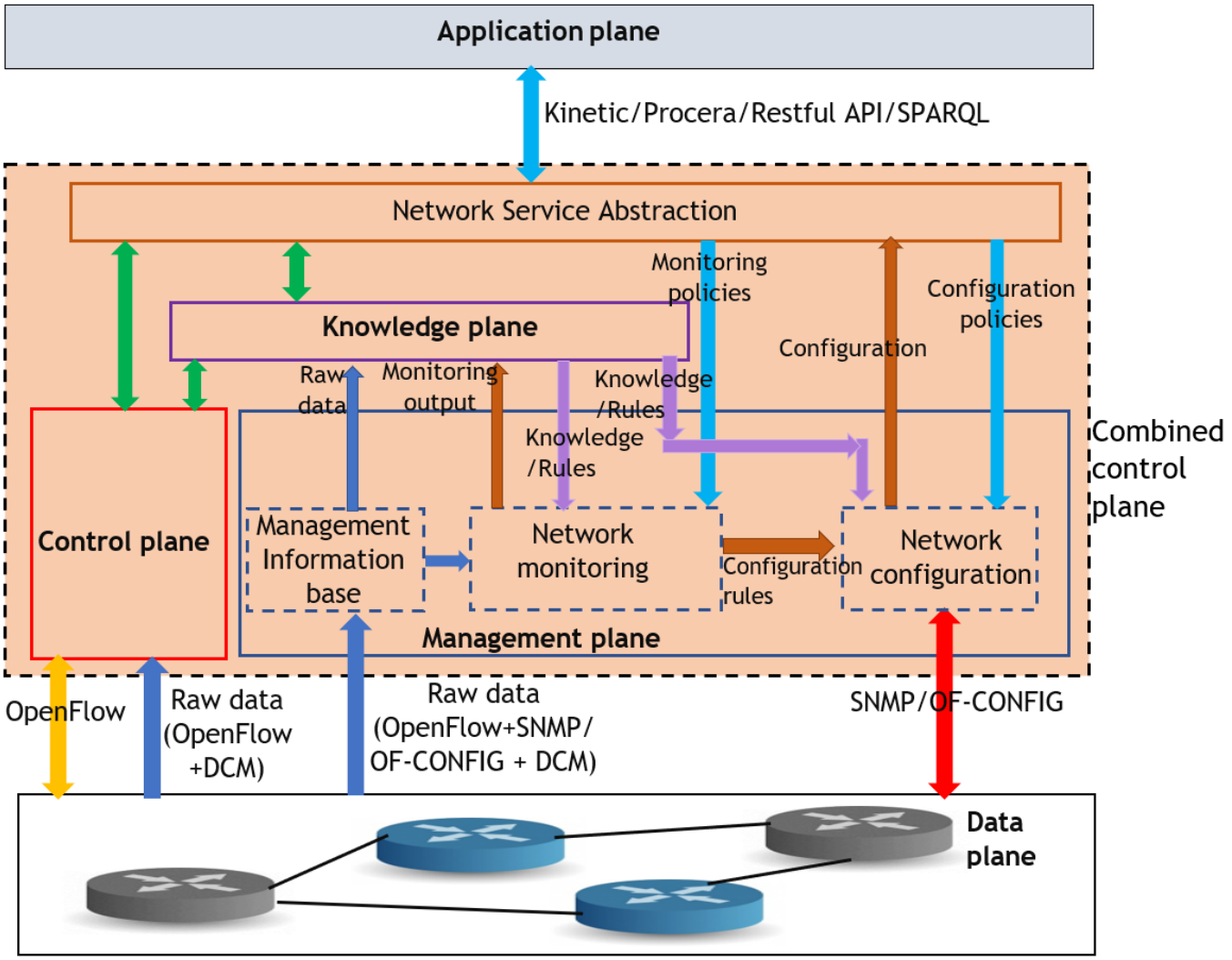

Figure 5. Architecture of the KDN management plane with detailed flows exchanged between management plane and other planes.

As shown in Figure 5, on the top of the architecture, there is an application plane for defining policies for making network control and management decisions, whereas, on the bottom, there is infrastructure (KDN or legacy switches). The Network Service Abstraction (NSA) layer is an interface for the control and management decision-making applications to connect with the rest of the architecture, which is an abstraction of the network services such as email services, file transfer, domain name systems, etc. [246]. The NSA layer provides a common interface for applications to communicate their service requirements to the knowledge, control, and management planes. The management plane (management abstraction layer) acts as an interface between the network services abstraction layer and the physical infrastructure of the network. Note that, in Figure 5, the control module works in parallel with the management abstraction layer.

The management abstraction layer provides a unified interface for data collection, device configuration, and monitoring of the underlying network, as shown in Figure 5. Raw data can be collected from the network and can be either directly used for network monitoring or provided to the knowledge plane for knowledge generation. The management information base is responsible for collecting and managing network data required for network monitoring tasks and knowledge generation. Note that raw data can also be collected by the control plane for either information-centric decision-making or to provide to the knowledge plane and use the generated knowledge or rules for decision-making. Data collected by each of the management and control planes have been discussed in Sections 4.5.2 and 5.4.1, respectively.

Network device monitoring can be either on-demand or continuous in nature. The output of a network monitoring task (monitoring output) can be provided to the knowledge plane for further knowledge/rule generation, as shown in Figure 5. The network monitoring module will generate configuration rules by comparing data or knowledge available with network monitoring policies [247]. These configuration rules will be provided to the network configuration module to directly configure the data plane elements, as shown in Figure 5. For example, consider a network energy-monitoring task. Assume that there is a network monitoring policy that specifies that "The average energy consumption of the network should be below 50%". Assume that the network monitoring module detects that "more than 50% of network nodes have an energy consumption greater than 90%", either using an information-centric approach or a knowledge-centric approach. Now, as the network monitoring output clearly indicates that the network monitoring policy is violated, it can create and issue a configuration rule to the network configuration module, such as "Reduce energy consumption of each node by 20%". Note that, in this example, monitoring output is "more than 50% of nodes have more than 90% energy consumption", which can be provided to the knowledge plane to further generate knowledge and use that knowledge to dynamically update energy management policies in the application plane. Note that, if network monitoring output is provided to the knowledge plane, further knowledge or rules on the monitoring output can be generated, and that output can have an effect on the application layer's policies. The management plane should frequently query the switches to collect the data it requires, such as flow statistics, network topology, configuration, etc. Monitoring accuracy and network overhead depend on the frequency of polling, where a higher frequency yields both high accuracy and high overhead. Note that, as evident from Figure 5, the network monitoring module can use data contained in the management information base for network monitoring or can directly collect data such as flow statistics, topology information, etc., using protocols such as SNMP. Network monitoring can be basically used for fault management, mobility management, and energy management tasks in KDN [248]. The functioning of these tasks is discussed in detail in the application plane in Section 7.3.2. Note that applications can convey network monitoring policies through the network service abstraction layer to the network monitoring module of the management plane, which defines procedures and guidelines by which the particular monitoring task should be carried out. Furthermore, the knowledge plane can have an effect on network monitoring by providing knowledge or rules to aid in network monitoring. Note that, in KDN, both information- and data-driven network monitoring and knowledge-driven network monitoring can occur. In data- and information-driven network monitoring, which is the conventional network monitoring approach in SDN, the network is monitored based on information and data collected from the network. In knowledge-driven network monitoring, raw data collected about the network parameters are provided to the knowledge plane to generate knowledge regarding the network status, which will use the generated knowledge for network monitoring to decide on actions to undertake considering network policies.

The configuration rules generated by the network monitoring module are provided to the network configuration module, which can be used to make decisions on network configuration considering rules or knowledge received from the knowledge plane and configuration policies received from the application plane. Note that the application plane can collect network configuration details, while the network configuration module can directly configure the network using a configuration protocol such as SNMP/OF-CONFIG, as shown in Figure 5. For instance, consider the previous example, where a configuration rule of "Reduce energy consumption of each node by 20%" is received from the network monitoring module to the network configuration module. Assume that there is a configuration policy specifying that "the minimum energy consumption of a node should be 5% if the average network throughput is greater than 5 Mbps; otherwise, the minimum energy consumption of a node should be 10%". Furthermore, assume that there is knowledge generated from the knowledge plane that specifies that the average throughput is 9.5 Mbps. Thus, by considering the available knowledge, the input configuration rule, and the configuration policy, the network configuration module can update the input configuration rules to output a new configuration rule to effectively reduce the energy consumption of each node. The updated rule for the given example should be "Reduce energy consumption of the node by a maximum of 20% subjected to the constraint that energy consumption of the node is not lesser than 5%". Note that the device configuration can occur iteratively for a few cycles until the network monitoring policy violation is lifted. More details on configuration rule generation and configuration policy enforcement are described in Section 5.4.3.

4.3. Network Management Protocols/Interfaces

In work [249], authors show that networks are hard to manage due to protocols exposing all their internal details, such that a Complexity Oblivious Network Management (CONMan) interface has been proposed, that includes minimal protocol-specific information that reduces the difficulty in configuration management. Similarly, Platform for Automated Operation and Configuration Management (PACMAN) achieves network configuration management using active documents and is designed to be operated by humans, which is prone to errors due to human errors [250]. Therefore, a software-defined management plane has been proposed to automate the task of network management. The OpenFlow Management and Configuration Protocol (OF-CONFIG), which relies on the Network Configuration Protocol (NETCONF) for transport, has been introduced for the management of resources in OpenFlow-enabled switches [251]. OF-CONFIG allows automatic discovery of switches, provides a standard way to configure the switches by installing flow tables and other settings, monitors the status of switches using traffic flow statistics, etc., and updates the switches. However, some have complemented the capabilities of OF-CONFIG/NETCONF while presenting a semantic-based approach that automates the configuration of network devices, where they have formalized the semantics of the switch configuration domain using a Web Ontology Language (OWL) and developed an ontology-based information extraction system for the Command Line Interface (CLI) of network devices [252]. They have further developed a learning algorithm to enable automated interpretation of the command-line interface's configuration capabilities in the network. This framework is highly compatible with the KDN, as the knowledge plane can dynamically create configurations in OWL format and provide them to the network configuration module in the management plane to directly convey the configurations to the switches, where the CLI will automatically extract the configurations from the ontology-based configurations. A framework for automated network management in hybrid KDN and legacy networks known as HybNET, considering the practical non-existence of fully KDN networks, is described in [253]. A protocol for monitoring and configuration of network devices in KDN known as Simple Network Management Protocol (SNMP) has been proposed that can be used as an alternative to OF-CONFIG/NETCONF [254]. SNMP provides a standardized way to remotely manage network devices, where network devices

act as servers that provide information about their status, performance, and usage to the centralized management client.

4.4. Network Monitoring Frameworks

A network monitoring framework known as Payless has been proposed for KDNs, which uses an adaptive network statistics collection algorithm at different aggregation levels using RESTful Application Programming Interfaces (APIs) to deliver highly accurate information in a timely manner with a lower communication overhead [255]. A Representational State Transfer (REST) API is an API that conforms to the constraints of the REST architectural style. OpenSample is a low-latency, sample-based measurement platform for network monitoring that has been designed to collect network load and flow statistics [256]. Another network monitoring framework called OpenNetMon monitors the network by collecting per-flow metrics to check whether end-to-end QoS parameters are actually met while delivering the per-flow metrics to the knowledge plane to compute paths using traffic engineering [257]. OpenNetMon also uses an adaptive rate for polling similar to Payless, where the rate increases when the flow rates differ between the samples and decreases when the flows stabilize. A joint HOst-NEtwork (HONE) traffic management framework is a traffic monitoring framework used in KDN using a collection of traffic measurement data that minimizes overhead by performing lazy materialization of fine-grained statistics and processes data locally on end hosts [258].

4.5. Network Data Collection

4.5.1. Data Collection Methods (DCMs)

Note that the management plane can collect data for network monitoring (flow statistics, network topology, performance metrics, etc.) and network configuration (configuration data) using a management protocol such as SNMP/OF-CONFIG or using a network monitoring framework. One of the main purposes of the management plane is to collect network data appropriately in order to generate knowledge using the knowledge plane. Furthermore, in the KDN paradigm, both network monitoring and network configuration functions are knowledge-driven rather than information-driven, even though an information-driven approach can be utilized, even in KDN. Thus, in a knowledge-driven approach, raw data also contain management-related data such as topology data, configuration data, performance metrics data, traffic flow data, etc., which are provided to the knowledge plane for knowledge/rule generation to drive network monitoring and configuration functions. As there are data types that cannot be extracted using one of the network management protocols discussed in Section 4.3 or the network monitoring framework discussed in Section 4.4, such as environmental data, detailed network traffic data, policy data, etc., which are used for knowledge generation or network monitoring, it is necessary to investigate Data Collection Methods (DCMs) to collect such data for the management plane. In other words, standard management protocols or network monitoring frameworks have not been designed to collect the customized data required for knowledge generation.

Data collection can occur in two modes. The first mode is the reactive mode, in which data from the infrastructure plane are pulled from the management plane reactively. The other mode is the proactive mode, where the nodes proactively send data to the higher planes without an explicit request from such planes. Usually, data are first collected by the nodes (devices), and then collected data are sent to the management plane by each node proactively [259]. However, this approach, which is usually used in the centralized architecture of KDN, is not the optimal way to collect data, as redundant data may be collected at the management plane, which yields higher communication costs, channel utilization, and latency. Therefore, a data collection optimization framework using Integer Quadratic Programming (IQP) has been proposed that minimizes total communication delay, communication cost, and communication overhead so that only a selected number of agents will unicast the collected data to the management plane while other nodes act as only broadcasting nodes [260]. Other techniques involve packet sampling techniques,

which have been proposed to sample wildcard flow entries to be collected as data [261]. An adaptive network data collection system that dynamically selects network data collection nodes based on network status and also samples network traffic based on flow characteristics to reduce the volume of data collection has been utilized in [262]. A data collection methodology to collect sensor measurements in an energy-efficient manner in a wireless sensor network environment of a KDN has been studied in [263]. Another data collection approach in vehicular network environments proposes to use a predictive data collection algorithm that cooperatively uses the cellular and ad hoc interfaces, using ad hoc interfaces where possible to limit the use of the cellular interface and reduce the communication cost at which routing decisions for such multi-hop data collection have been realized based on the network status [264].

4.5.2. Types of Data

The management plane collects data to gain insight into the behavior of the network and its users in order to make decisions about network configuration and optimization. The management plane may collect different types of data related to network management from the infrastructure plane. The data collected and stored in the management plane is the basis for knowledge generation. Different types of data, such as topology, configuration, traffic flow, event logs, resource usage, performance metrics, sensor data, etc., can be collected by the management plane.

It can collect configuration data related to the configuration of the network devices, such as the operating system, device name, network interface addresses, routing protocols, security settings such as access control lists, and quality of service settings [265]. The collection of configuration data is useful for configuring the network using the management plane.

Further, the management plane may collect devices' resource usage data, such as CPU utilization, memory utilization, network interface statistics, etc., in order to identify potential performance issues and optimize network performance [266].

In addition, the management plane can collect data about network traffic flow, including the source and destination of traffic, the amount of traffic, and the protocols used, in order to form a global-level traffic matrix at the management plane [267]. These data can be used to identify patterns in network usage and can be further analyzed using the knowledge plane for energy management, mobility management, and fault management. For instance, network traffic data can be analyzed to identify usage patterns of energy consuming devices to aid in optimizing energy usage and facilitate proactive maintenance of network infrastructure by identifying potential failures. Note that network traffic flow data are collected jointly by the control and management planes.

Further, the management plane may collect event logs, which include information about the events that occur in the network, such as system errors, security alerts, configuration changes, etc. [268]. These event logs can be used for network troubleshooting.

Another piece of data that can be jointly collected by both the management and control planes is the network topology information, which includes the location of the switches, routers, and other network devices in order to develop a global view of the network [269].

Furthermore, the management plane may collect data from sensors such as temperature, pressure, humidity, etc. for real-time predictive fault diagnosis of communication infrastructure [67].

The management plane monitors and allocates resources for network devices. Specifically, the management plane involves fault, mobility, and energy management. Thus, it collects performance metrics such as energy consumption, device mobility metrics (position, velocity, acceleration), and device availability/failure rate [270].

Furthermore, the management plane can collect configuration policies and monitoring policies related to monitoring tasks for device energy management, mobility management, fault management, and configuration tasks in order to verify the policies. Monitoring policies usually involve policies for resource allocation, troubleshooting, and analysis [271].

4.5.3. Data Representation Models

Data models provide a standard and consistent way of modeling network data, which makes it easier to automate and manage networks. Yet Another Next Generation (YANG) is a modeling language used to describe the state data of network devices and their configurations. The NETCONF and YANG data formats are often used together to offer a standardized method for configuring and controlling network devices. Network administrators may use the YANG data model to define the data that are used to configure network devices and to make sure that the data are consistent between various devices and suppliers [244].

Another data model called the Common Information Model (CIM) offers a uniform way to express data about network infrastructure and device services. A collection of standardized object classes and characteristics that define different facets of network infrastructure and services serves as the foundation for using the CIM data model in KDN. These object classes and properties provide a common language for different networking devices and systems to exchange information about their configuration, status, and performance [245].

Table 4 summarizes management protocols, network monitoring frameworks, and data collection methods in the management plane.

Table 4. Summary of management protocols, network monitoring frameworks, and data collection methods in management plane.

Category	Protocol/Framework/Method	Purpose	Developer
Network management	CONMan [249]	Complexity oblivious network management with minimum protocol specific data	H. Ballani et al.
	PACMAN [250]	Network configuration management using active documents	X. Chen et al.
	OF-CONFIG [251]	A protocol for configuring and monitoring OpenFlow switches	R. Narisetty et al.
	Ontology based configuration [252]	Automatic configuration of switches using an OWL-based ontology	A. Martinez et al.
	HybNET [253]	Automated network management framework in hybrid KDN and legacy networks	H. Lu et al.
	SNMP [254]	A protocol for monitoring and configuration of network devices	J.D. Case et al.
Network monitoring frameworks	Payless [255]	Network monitoring with adaptive network statistics collection	S.R. Chowdhury et al.
	OpenSample [256]	Sample based network load and flow statistics monitoring	J. Suh et al.
	OpenNetMon [257]	Monitoring using adaptive rate polling to collect per flow metrics	N.L.V.Adrichem et al.
	HONE [258]	Traffic monitoring using traffic measurement data	P. Sun et al.
Data collection	IQP [260]	Data collection minimizing communication cost, delay, and overhead	S.N. Wijesekara et al.
	Packet sampling [261]	Collect sampled wild card flow entries	P. Wette et al.
	Adaptive data collection [262]	Dynamically selects data collection nodes and samples network traffic based on flow characteristics	D. Zhou et al.
	Sensor measurements collection [263]	Collects sensor measurements in an energy efficient manner	W.H. Liao et al.
	Predictive data collection [264]	Predicts and uses multihop routing using ad hoc and cellular interfaces for data collection	Z. Jiao et al.

5. Control Plane

5.1. Introduction to Control Plane

The control plane in KDN is logically centralized and self-learning with respect to knowledge, where the control is based on both application policies and knowledge learned or rules derived from network data. As knowledge and rules are generated based on real-time data, adaptive control decisions can be made, such as identifying congested links based on knowledge of network traffic flow to direct traffic through less-congested

links. This controlling approach is more efficient than the traditional SDN approach, where network policies determine network control and knowledge-based decisions are not made.

5.2. Interfaces and Sub-Planes Connected with the Controller

The low-level architecture of KDN with sub-planes and interfaces between the planes [272] is shown in Figure 6.

Note that, as shown in Figure 6 and as explained in Section 2, the control plane, the management plane, and the knowledge plane can be abstracted to a combined control plane [62]. This combined control plane in KDN additionally generates knowledge and rules, and the controller's actions are influenced by knowledge and rules compared to the controller in SDN.

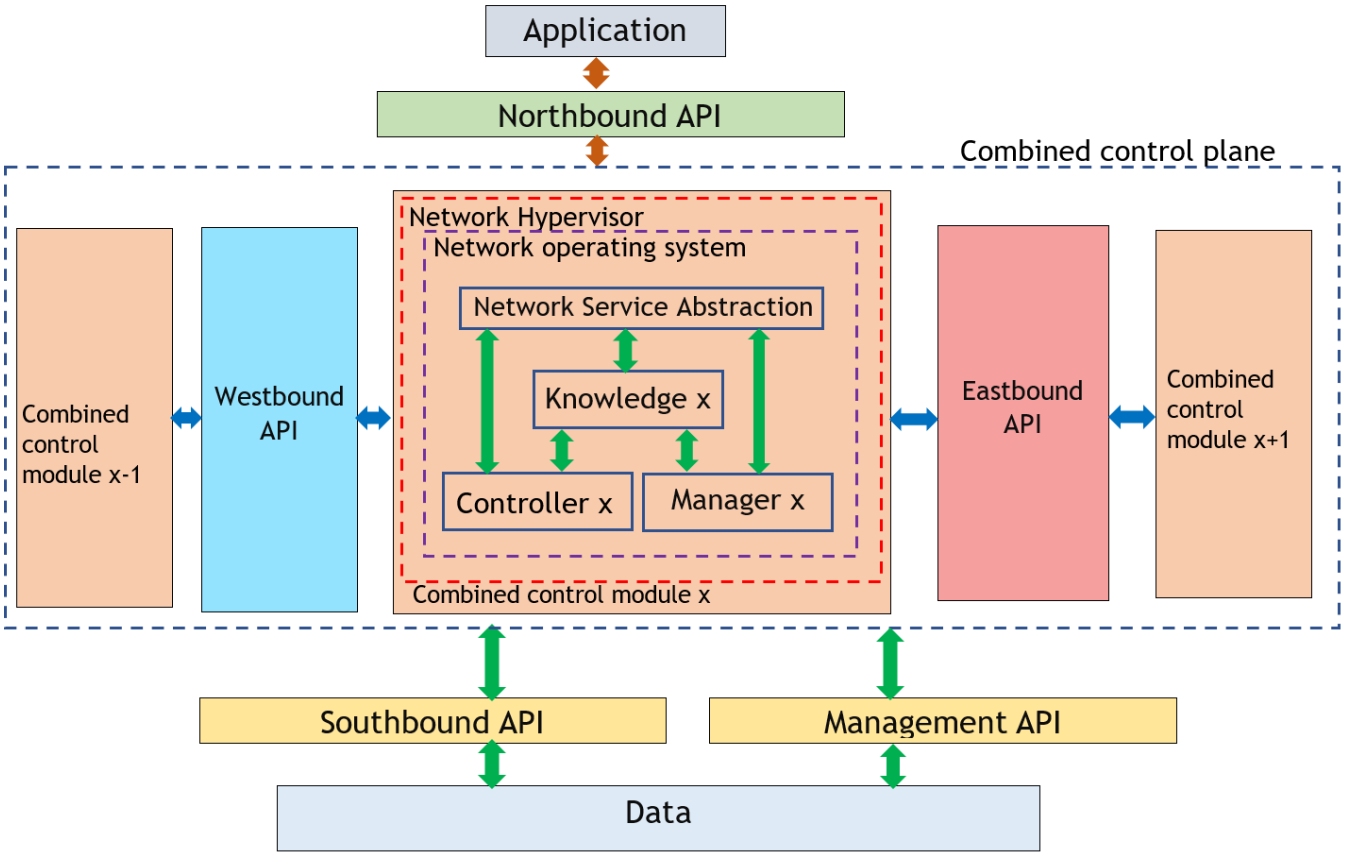

Figure 6. Architecture of KDN with sub-planes and interfaces between the planes.

5.2.1. Northbound Interface

The northbound API enables communication between the combined control plane and the application plane. Thus, the northbound API in KDN communicates with all management, knowledge, and control planes through the network service abstraction layer. The network service abstraction layer allows applications to communicate their network service requirements to the management, control, and knowledge planes. Furthermore, by providing a standardized interface for application control of network resources, the network service abstraction layer enables network service providers to develop and deploy new services more easily. An interface that supports application portability and interoperability among different control platforms is much desired where the programming language and controller are independent in nature. In other words, the northbound interface hides the complexity of the underlying combined control logic from the application plane. Unlike the OpenFlow protocol, which is the dominant protocol used for the southbound interface in KDNs, a single northbound API does not stand out from others, as the requirements of

different applications are different from each other and different data models are used in different control frameworks.

Firstly, there are in-built controller-specific ad hoc northbound interfaces. NOX has a functionality-based ad hoc northbound interface that does not consider the safety of network applications [273]. With the aid of parallel asynchronous event processing, the northbound interface in ParaFlow provides concurrent execution of multiple applications [274]. The northbound ad hoc interface for the Rosemary controller is secure, as it provides application containment and an application permission structure that prevents the controller from stumbling in the case of application failure [275]. Application containment in the northbound interface in Rosemary is achieved by spawning applications separated by a micronetwork operating system, while the application permission structure involves a sandbox approach for access control and authentication of applications.

Secondly, controllers such as FloodLight, OpenDayLight, etc. have RESTful-APIs for the northbound interface based on the REST architecture. In REST architecture, communication takes place as stateless client-server communication, where client information is not stored between get requests and each request is separate and unconnected [276]. OpenDayLight has a model-driven service abstraction layer that allows integration of network services requested by the application layer through the northbound interface. The applications can configure and retrieve network information through the RESTful-API in OpenDayLight [277]. FloodLight has RESTful-APIs for obtaining and setting the controller's state and contains Java event listeners for event notification and passing emitted events. The northbound API in FloodLight enables communication between an asynchronous, event-driven, modular, and Java-based application framework and the controller [278].

Thirdly, there are intent-based northbound APIs for the applications to specify policy-based directives, which are converted to forwarding rules by the control plane and communicated to the data plane. The network intent composition in the OpenDayLight (ODL) controller allows external applications to give directives to the ODL controller using intents [279]. Similarly, the Open Network Operating System (ONOS) distributed controller also has an intent framework for applications to specify requirements as network policies [280]. However, only a few controllers, such as ODL and ONOS, support intent-based APIs despite their high flexibility and application portability.

Fourthly, some researchers have attempted to build northbound APIs for communication between the application plane and the control plane. SFNet is such an API that provides high-level primitives with which network applications can request resource reservations and verify network status using JSON messages, where access is granted based on resource availability [281]. TinyNBI is a northbound interface proposed to address portability issues in different OpenFlow versions using a language-independent low-level interface. In tinyNBI, there are abstractions defining capabilities, configuration, and statistics that represent features of OpenFlow versions, where missing abstractions are handled either by offloading or providing an error message [282].

Finally, there are a set of programming languages that can be used to write northbound APIs in order to achieve the objectives of the northbound interface specified earlier. A high-level programming language can be used to transfer application requirements into packet forwarding rules. Common configuration languages such as Command Line Interface (CLI) offer primitive abstractions for hardware configurations. New high-level languages to interface with the controller that have emerged for configuring using the CLI are error-prone and require a lot of effort. A programming language for KDN can be used to achieve several high-level tasks such as flow installation using either reactive or proactive approaches, static or dynamic policy definition, and network abstractions such as flow matching, security, traffic engineering, etc.

Flow-based Management Language (FML) is the very first KDN-based programming language, known earlier as Flow-based Security Language (FSL), which is a high-level declarative policy language based on logic programming [283]. This language allows the writing of policies and provides abstractions to allow or deny flows, redirect flows to pass through a given host, and specify flow bandwidth. Nettle, being one of the early programming languages for KDN, supports event-based and time-varying operations, allowing it to create dynamic policies that are based on functional reactive programming [284]. Procera, a KDN programming language, is also based on functional reactive programming, which is an extension of Nettle, which has a set of abstract data types and signal functions to describe policies by applying windowed histories to events [285]. Frenetic is the programming language from which most of the current KDN programming languages have been derived. It has a functional reactive combinator library to describe high-level packet forwarding policies and a Structured Query Language (SQL)-like language for classifying and aggregating traffic of the network [286]. Kinetic is a KDN programming language evolved from Frenetic, which expresses policies in terms of finite state machines, where states represent distinct forwarding behaviors and events trigger transitions between the states [287]. It can encode a generic finite state machine that is applied to a packet group, where each group has a separate state machine instance. PonderFlow is an extension of Ponder language designed for specifying policies for OpenFlow-based networks that also provides access control and implements authorizations and obligations. Obligations in PonderFlow define what actions are to be performed by the controller when certain events occur in the network [288]. Maple is a programming language consisting of an optimizer and a scheduler, where the optimizer uses a trace tree data structure to store the invocation of the algorithm on a particular packet and then generalizes rules in the flow table, whereas the scheduler applies parallelism on servers binding the controller's thread to a client switch [289]. Merlin is a network programming language derived from Frenetic that has been designed to implement network policies and also provides mechanisms to check whether sub-policies violate global constraints or not. Merlin also provides constructs to provide bandwidth limits and guarantees [290].

Table 5 summarizes the details of programming languages that can be used to achieve high-level tasks in KDN.

Table 5. Summary of programming languages in KDN.

KDN Language	Paradigm	Policy Definition	Flow Installation	Network Monitoring Policy	Birth Year	Base Language
FML [283]	Declarative (Logical)	Static	Reactive	Access data collected from other resources	2009	C++, Python
Nettle [284]	Declarative (Functional reactive)	Dynamic and static	Reactive	N/A	2010	Haskell
Procera [285]	Declarative (Functional reactive)	Dynamic and static	Reactive	Data-Windowed history, other resources	2012	Haskell
Frenetic [286]	Declarative (Functional reactive)	Dynamic and static	Reactive	Data-Windowed history, Query language	2010	Python
Kinetic [287]	Declarative (Event driven)	Dynamic and static	Reactive, proactive	Data-Windowed history, other resources	2014	Python
PonderFlow [288]	Declarative (Event driven)	Dynamic and static	Reactive	N/A	2014	Java
Maple [289]	Imperative	Dynamic and static	Reactive, proactive	Access information collected from other resources	2013	Python
Merlin [290]	Declarative (Functional)	Dynamic and static	Reactive	Windowed history of data	2013	OCaml

5.2.2. Southbound Interface

The southbound API is the interface between the data plane and the control plane, which is used to transfer data from the forwarding elements to the controller and control from the controllers to the data plane elements. This communication takes place with the help of standardized protocols.

OpenFlow is the most widely used protocol for the Southbound API standardized by the Open Network Foundation (ONF), which is non-vendor-specific, allowing OpenFlow-enabled devices from different vendors to be interoperable [291]. OpenFlow standardization is driven by the decoupling of the control and data planes. It provides a common specification to implement data plane devices and the communication channel between the data plane and the control plane. There are different data flows from the data plane to the control plane that are realized using OpenFlow. In an OpenFlow architecture, the switch necessarily contains a flow table and an abstraction layer. A flow table element consists of rules to match incoming packets (destination and source IP addresses, destination and source MAC addresses, destination and source port addresses, etc.), a set of instructions/actions to undertake upon receiving a match, and statistics for the particular flow such as the number of packets, flow duration, number of bytes, etc. When a packet is received at an OpenFlow switch, it will extract the packet header and look for a matching entry in the Flow table. If a matching entry is found, corresponding actions will be executed; otherwise, corresponding actions for the table-miss flow entry will be carried out. The action for a table-miss flow entry can be to drop the packet, search in the next flow table, or forward it to the controller. A packet processing pipeline was introduced in OpenFlow 1.1, which can implement multiple flow tables per switch. When the switches are unaware of forwarding actions for a matched incoming traffic flow or in the case of a table-miss flow entry (when the action of the flow table is "forward to controller"), "packet-in" messages are sent from the switches to the controller. Then the controller will determine the forwarding path and send the forwarding rule and action back to the switches in the form of a "Flow-Mod" packet. When the controller sends flow table updates (Flow-Mod) for a switch in response to a packet-in message, it is a reactive approach, whereas, when the controller sends updates for a flow table without a packet-in message, it is a proactive control approach. Furthermore, the controller collects flow statistics of forwarding devices using "Switch Feature" messages and event-based messages such as link or port changes of a switch using "CPort-status" messages. OpenFlow has a meter table in order to support QoS in KDN, which comprises multiple meter entries. Recent versions of OpenFlow have introduced the bundle concept, which allows performance of modifications on a group of forwarding switches. OpenFlow uses Transmission Control Protocol (TCP) as the transport layer protocol. Even though OpenFlow is the most widely used Southbound API protocol, there are many other alternatives to it.

The main alternative protocol for OpenFlow is Forwarding and Control Element Separation (ForCES), which also defines the control and data elements separated from the network devices [292]. However, the difference compared to OpenFlow is that the control plane and the data plane are kept close to each other (such as within the same network device) without the need for a logically centralized external controller. Thus, ForCES allows network programmability without changing the conventional network architecture. This protocol works in master–slave mode, where the control element is the master and the forwarding element is the slave. The forwarding elements consist of logically functioning blocks that perform a specific function. The control elements instruct the forwarding elements on how to forward packets using logically functioning blocks. ForCES uses Stream Control Transmission Protocol (SCTP) as the transport layer protocol, which is more reliable and secure than TCP. However, ForCES is less utilized in KDN compared to OpenFlow, as ForCES lacks open-source support.

OpFlex is another southbound API with the intention of distributing part of network management back to the data plane in order to improve scalability. Policies are logically centralized and abstracted from the data plane, where they are stored in a repository and communicated to policy elements (data plane elements). It also has an observer repository for storing network events and faults. However, OpFlex lacks network programmability and dynamic network control [293].

Protocol Oblivious Forwarding (POF) makes the forwarding plane oblivious to the protocol by using a generic Flow Instruction Set (FIS), where the forwarding element does not need to know about the packet format [294]. In POF, forwarding elements are under the full control of the controller, which will install keys and table lookups in the switches. Thus, in POF, the packet header field is not required to be inspected in packet forwarding, which speeds up packet forwarding, unlike in the OpenFlow protocol.

Another promising protocol for the southbound API is the Path Computation Element (PCE) protocol [295]. In this protocol, the Path Computation Client (PCC) in the data plane requests forwarding paths from the PCE residing in the control plane, which computes paths with the aid of a traffic engineering database.

OpenState attempts to extend OpenFlow by extending OpenFlow match-action abstraction to extended finite machines, which is an attempt to decentralize some control back to forwarding plane elements [296]. In OpenState, finite state machines facilitate several stateful tasks inside the switches, where all tasks that involve local state are carried out without explicit control instructions from the controller.

5.2.3. East- and Westbound Interfaces

A single network centralized controller has the possibility of failure or a tendency for poor performance when interacting with a large number of user equipment. Furthermore, under a single controller, malfunctioning of the controller can lead to a single point of failure. Thus, a single control architecture has scalability and reliability issues. The solution proposed for that is a logically centralized control plane with physically distributed multiple controllers communicating with each other for synchronization using East and West bound interfaces. These interfaces should provide common compatibility and interoperability between different controllers and need to coordinate flow setups originated by applications, exchange reachability information, and update it to keep the network consistent. The main purpose of these interfaces is to provide interaction between the physically distributed controllers in order to have a global view of the network. As interactions, data can be exchanged between the controllers; they can observe/notify their capabilities; and they can provide procedures for data steadyness models.

Application Layer Traffic Optimization (ALTO) is an east- and westbound API used to optimize point-to-point traffic and provide guidance for peer selection. ALTO provides guidance for peer selection by having information such as the location and characteristics of all nodes in the network [297].

Hyperflow has an east- and westbound API that is based on an event propagation system that publishes a change in a network domain controlled by a given controller to other controllers whenever a change is detected, using a publish/subscribe system [298]. For instance, if a controller failure is discovered, it will be published to other controllers so that affected data plane elements will be handed over to a nearby controller, thus improving the availability of the system. Furthermore, HyperFlow is resilient to network partitioning by continuing to operate independently. Similar to Onix and ONOS, every controller in the physically distributed control plane of HyperFlow has a global network view.

ONOS also has east- and westbound interfaces for physically distributed controllers, which will protect the network from controller failures by connecting a given data plane element with multiple controllers, where one will be the master controller and others will be the backup controllers, using east- and westbound interfaces [299]. In the event of a master controller failure, one of the backup controllers will take over. Furthermore, ONOS has additional recovery protocols to recover from lost crashes due to controller updates.

Similar to ONOS, Onix also has each controller responsible for a subset of the network; however, it has a global network view where scalability is ensured using network partitioning and aggregation, and communication between the controllers is achieved using east and westbound interfaces [300]. Onix divides the network's information base such that each controller is responsible for a portion of it while aggregating the data using applications to lower the information fidelity.

Note that east- and westbound interfaces are absent in an architecture that is both logically and physically centralized.

5.2.4. Management API

The management API acts as an interface between the management layer and the data plane, which uses protocols such as SNMP, OF-CONFIG/NETCONF, etc. for communication as described in Section 4.3.

5.2.5. Network Hypervisors

A network hypervisor's primary function is to provide virtualization, which enables the installation of different network operating systems on a single physical server. By allowing several virtual computers to share the same hardware resources, hypervisors perform this. As a result, hypervisors effectively cut the cost of the infrastructure. Virtual machines' ability to be moved, created, or destroyed on demand also makes it possible to offer elastic services. The network infrastructure must be able to accommodate any network topologies and addressing methods in order to allow complete virtualization. However, it is practically challenging to realize such infrastructure. Topology and address virtualization are made possible by network virtualization, which also enables resource provisioning, management, and monitoring over virtual networks.

With the use of an abstraction layer to slice data based on ready-made OpenFlow-capable switches, FlowVisor enables many logical networks to share the same OpenFlow networking infrastructure, enabling the coexistence of numerous and heterogeneous networks. Traffic, CPU, forwarding tables, bandwidth, and topology are the dimensions taken into account for slicing in FlowVisor. Each network slice supports a controller, allowing several controllers to coexist in the same physical network while each controller is free to operate on its own network slice [301].

Another network hypervisor called OpenVirtex creates virtual networks using control functions, topology, and address virtualization, which is similar to FlowVisor, which acts as a proxy between the network operating system and forwarding devices [302].

While FlowVisor is a full virtualization technology, FlowN provides container-based virtualization that is designed to be scalable and provides the opportunity to share a controller to manage multiple domains [303].

HyperFlex decomposes the hypervisor into functions for virtualizing KDN networks and also has a control plane isolation function for control plane virtualization such that control plane resources are correctly shared between each virtual KDN network while, at the same time, protecting the resources from exhaustion [304].

TeaVisor is a network hypervisor that guarantees bandwidth isolation by tackling the overloaded link problem, which has three components: path virtualization, bandwidth reservation, and path establishment [305].

VMware is another network virtualization platform where network functions are embedded in the hypervisor and distributed across the environment, which provides the entire network model in software. This network virtualization platform allows the creation of virtual networks, each with its own service model and topologies, addressing architectures having the same physical network [306].

5.2.6. Network Operating System

A network operating system should provide abstractions to access and interact with lower-level devices, provide the ability to access resources such as hard drives, network adapters, CPUs, etc. concurrently, provide services, and provide security mechanisms. The common services offered by a network operating system are providing network state and network topology information, device discovery, and the distribution of network configuration. Thus, the core functions provided by a network operating system are topology management, device management, notification management, statistics management, routing, and security mechanisms in conventional SDN architecture. However, in KDN architecture, a network operating system will additionally have the functions of knowledge creation and knowledge management. As a result, in the KDN design, the network service abstraction layer requires all of these services to be provided by either the control plane, management plane, or knowledge plane. As shown in Figure 6, a network operating system offers a platform for the implementation of the knowledge plane, the management plane, the control plane, and the network service abstraction layer in the KDN architecture. Keep in mind that, in the KDN design, the management abstraction layer is responsible for all management-related activities, while the controller is in charge of other duties such as routing and security, and the knowledge plane is in charge of knowledge creation and administration. However, in the original SDN architecture, all tasks of the network operating system were assigned to the controller. Before the advent of SDN, networks were managed using proprietary network operating systems and device-specific instruction sets. The purpose of a network operating system is to abstract device specific characteristics and provide common functionality. Due to network operating systems, application developers do not need to know about the low-level details of data distribution among forwarding elements; they only need to specify policies, which will be converted into low-level details by the network operating system and southbound APIs. The network operating system resides inside the combined control plane, which works according to the policies defined by the applications to provide knowledge-driven control and management operations to the data plane in the KDN paradigm.

Network operating systems are discussed in detail with respect to the distribution of the control plane in Section 5.3.1.

5.3. Control Models

5.3.1. Centralized vs. Distributed vs. Hybrid Control

Even though the originally proposed SDN architecture is logically and physically centralized, having strengths such as programmability and flexibility introduced into the network, due to difficulty in handling large networks and network failure under single controller failure, other distributed control models have emerged. The control can be categorized based on its distribution, as given below.

- Logically and physically centralized control model;
- Logically centralized and physically distributed with flat control model;

- Logically centralized and physically distributed with hierarchical control model;
- Logically and physically distributed control model;
- Hybrid control model.

Each of the above control models is discussed in detail in the following sections. Each of the control models, based on the degree of distribution of the control plane, is graphically illustrated in Figure 7.

Data Plane Element

In order to implement different control architectures, a forwarding plane node should have the components shown in Figure 7f. The KDN module performs packet processing with the help of control messages received from the control plane, while the KDN module configures the switch using the configurations received from the management plane. Furthermore, the KDN module collects data to be sent to management and control planes. The local control agent will receive control information, and the local control agent has the passive action of just forwarding the received control messages from the KDN controller to the KDN module in the node when the control channel is available. Note that, in KDN, the control channel refers to the communication channel that exists between the switch and the control plane, while the management channel refers to the communication channel that exists between the switch and the management plane. Thus, the local controller is an optional element that can be considered unavailable or not implemented in pure KDN switches. However, in hybrid control architecture, when communication with the centralized controller is lost, the local control agent will act as an active backup controller, which can switch to traditional ad hoc routing protocols, where the control information will be exchanged between the nodes. Thus, in hybrid KDN switches, the local controller needs to be implemented. The normal operation of a switch/node which is sending/receiving data and forwarding packets sent by other nodes can be achieved using the data channel.

Logically Centralized and Physically Centralized Control

The core concept of SDN architecture, having a logically centralized control plane, is secured in this architecture. Figure 7a shows the communication in a logically and physically centralized control architecture. As evident from Figure 7a, this architecture preserves the centralized SDN architecture, where all actions performed by the nodes are explicitly defined by the SDN controller. This architecture is the simplest and easiest to manage. Even though this architecture has the highest level of flexibility and programmability, it has a higher delay due to the long-distance communication between the nodes and the centralized controller. In this model, there is a single logical and physical controller that is decoupled from the data plane. A single controller is responsible for controlling all forwarding devices of the network, which is a perfect model in terms of simplicity. However, in this model, the controller's performance degrades when the network becomes large, where controller bottlenecks can occur when dealing with large numbers of requests from forwarding elements and can struggle to provide the same performance as in smaller networks. Specifically, data center networks, having thousands of switching elements, have overloaded the controller and shown a bottleneck in throughput, while service provider networks (wide area networks), having geographically distributed nodes with a large network diameter, have caused high latency in the controller [307]. Furthermore, the controller can become a single point of failure, so reliability is low in this model.

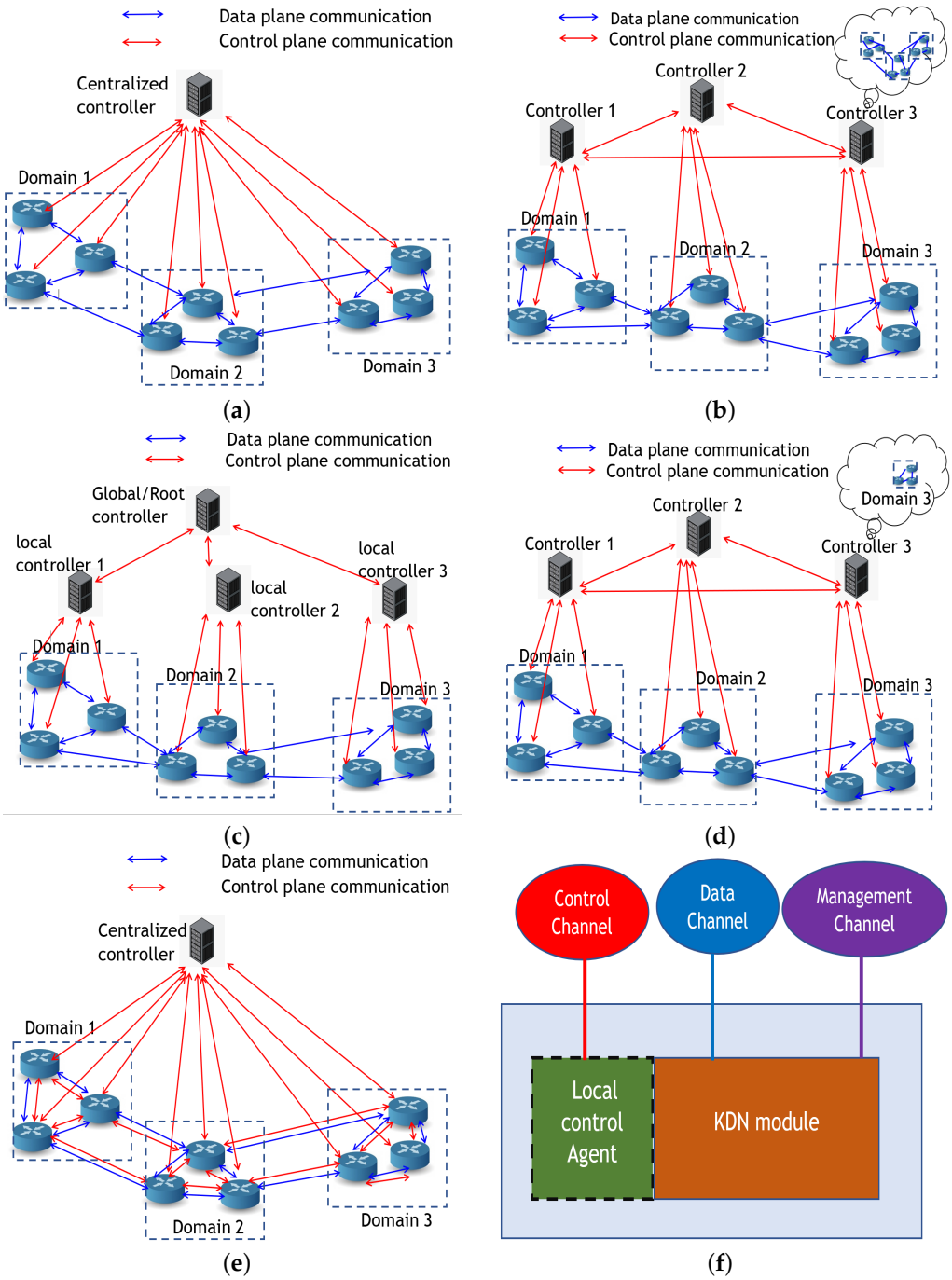

Figure 7. Different control models based on degree of distribution of the control plane and structure of a data plane element. (**a**) Single controller (Logically and Physically centralized) [273]. (**b**) Logically centralized physically distributed with flat control [298]. (**c**) Logically centralized physically distributed with hierarchical control [308]. (**d**) Logically and physically distributed control [309]. (**e**) Hybrid control [310]. (**f**) The structure of a node in KDN [311].

The Network Operating System (NOX) is the earliest network operating system that was designed for a physically and logically centralized control architecture with an event-based programming model, which has shown limited performance in terms of throughput [273]. NOX-MT (NOX-Multi Threaded) is an improved version of NOX that uses multi-threading and optimization techniques such as I/O batching to promote threshold performance [312]. Beacon is a Java-based open-source network operating system that has been designed to provide the runtime ability to start and stop existing and new

applications with high performance [313]. FloodLight is a flexible and easy-to-expand Java-based network operating system created based on Beacon, which has high CPU efficiency but requires more memory [278]. FloodLight-based controllers have shown vulnerability to DDoS attacks, so a secure NOS called SE-FloodLight incorporating secure features has been proposed [314]. All network operating systems discussed above (NOX-MT, Beacon, and FloodLight) take advantage of parallel processing using multi-threading capabilities in multi-core computers to implement concurrent systems in order to achieve the high throughput requirements of networks.

Trema is an open-source OpenFlow controller framework that allows easy implementation of arbitrary network control applications, specifically for data centers [315]. RYU is an open-source network operating system that is designed to expand the deftness of the system by using simple traffic control that supports OpenFlow, NETCONF, OF-CONFIG, etc. [316]. RYU executes different functionalities using events and handlers that are utilized to actualize synchronization between application calls and the RYU controller. Meridian is a platform for having a service-level model for application networking in clouds that integrates application provisioning in the cloud with the network using programmable interfaces [317]. Rosemary is a robust, secure, and high-performance network operating system that has a network application containment and resilience strategy when spawning applications [275]. Rosemary NOS has procedures for process containment, resource utilization monitoring, and an application permission structure to prevent network application failure leading to network control failure. As a result, Rosemary NOS's major objective is to secure and isolate applications through the use of a container-based architecture. The Rice University-developed Maestro is a different physically and conceptually centralized network operating system that is based on Java and offers APIs to construct modular network management apps to access and change network information [318]. ParaFlow is a C++-based multi-threaded centralized controller that makes use of parallelism in event processing by event handlers and uses mutex-based synchronization for consistency. It has a flow-based programming interface, where application programs can be developed based on network flows [274].

Logically Centralized and Physically Distributed with Flat Control

In a logically centralized and physically distributed flat control architecture, the network is horizontally partitioned, where each partition is controlled by different controllers and there is no hierarchy (flat) between the controllers, as shown in Figure 7b. Even though scalability issues in a single controller architecture are reduced by a flat hierarchy of controllers, when the number of controllers is limited and fixed under very large networks, scalability issues can still exist, as a given controller can handle a fixed number of user equipment without having a performance bottleneck.

SMaRtLight is a fault-tolerant controller platform with a logically centralized flat hierarchy master–slave mode of operation [319]. In SMaRtLight, a master controller is in charge of the network, backed up by slave controller replicas synchronized with the master to take over master control in case of master controller failure. Replicated state machines and a shared data store are used to achieve consistency between the master controller and slave controller replicas.

In HyperFlow, every controller in the physically distributed control plane has the global network view, where a change in a network domain controlled by a given controller is published to other controllers whenever a change is detected, such as controller failures, such that affected data plane elements will be handed over to a nearby controller [298].

ONOS has a master–slave working principle, where it will protect the network from controller failures by connecting a given data plane element with multiple controllers, where one will be the master controller and others will be the backup controllers. In the event of a master controller failure, one of the backup controllers will take over [280].

Onix uses a network information base data structure to store global network state, which is synchronized between other Onix controllers. Onix also has each controller, which is responsible for a subset of the network, having a global network view, where scalability is ensured using network partitioning and aggregation [300].

Ravana is a logically centralized and physically distributed fault-tolerant controller platform that uses replicated state machines that are extended to switch-side mechanisms to process control messages in order and exactly once, even during failures. In the Ravana framework, the network programmers need to write applications for the main controller, where the Ravana master–slave protocol will ensure replication of control logic to backup controllers such that, when the master crashes, a slave controller will be elected to take over the affected switches [320].

Distributed control systems should also have consistency, which means that data updates on distinct nodes should be updated on the controller nodes. Some distributed control systems have weak consistency, which means that updates in the data plane will eventually be updated in the control plane. Weak consistency can cause different controllers to read different values of the same property over a period of time. HyperFlow has weak consistency compared to other distributed control operating systems.

Logically Centralized and Physically Distributed with Hierarchical Control

In a logically centralized and physically distributed hierarchical control architecture, the network control plane is both horizontally and vertically partitioned. Note that vertical separation results in a hierarchy of controllers, where some controllers are controlled by one or more other controllers. We can find local controllers at the bottom of the hierarchy and global controllers at the top, as depicted in Figure 7c. The local controllers deal with local applications' requirements with frequent events, whereas the global controllers deal with non-local applications requiring a global network view with rare events. Due to the global/root control having a global view of the network, the control is logically centralized. This is different compared to flat-hierarchy distributed control, as, in that architecture all controllers can have a global view of the network, as opposed to local controllers, which have only a local view of the network in the hierarchically distributed architecture. However, even though a physically distributed hierarchical control architecture provides a solution to scalability, reliability, and vulnerability issues in a single controller, this architecture can suffer from path stretch problems [321].

Kandoo is one of the very first proposed hierarchically distributed control frameworks, having two layers of global and local controllers that partition control applications into global and local [308]. Local controllers, which are not connected to each other and have local network views, handle frequent events that occur in the data plane that do not require a global view, while the global controller (root controller), which is connected to all local controllers, handles non-frequent, non-local events that require a global view. Thus, Kandoo effectively reduces the events received by the root controller by distributing the load across multiple local controllers for frequent events. Even though the Kandoo framework has addressed the scalability issue present in logically and physically centralized control architectures, it does not have the fault tolerance to protect itself from failures and attacks.

Orion is a framework designed to overcome the scalability issue of flat control architectures and the path stretch problem in hierarchically distributed control architectures. In Orion, a hybrid hierarchical architecture is proposed to reduce the complexity of the control plane, while an abstracted hierarchical routing method is used to address the path stretching problem [322]. Orion also has two control layers, where the bottom layer consists of area controllers and the top layer consists of sub-domain controllers that have a global network view for their own domain and communicate with each other for synchronization.

FlowBroker is a hierarchical control architecture having domain controllers and one or more super controllers (brokers), where each domain controller can attach to more than one broker based on their reputation for load balancing and reliability [323]. In FlowBroker, brokers can also communicate with each other to share abstracted network states.

B4 is a framework that is employed in a software-defined wide area network that has a two-level hierarchy of the control plane [324]. The bottom control layer has Onix-based controllers, which are responsible for local site-level control, where each of the site-level controllers is managed by a global-level controller at the top level. Thus, there is a logically centralized controller that can implement high-level control with a global controller view without having a performance bottleneck with large networks. B4 has many fault-recovery mechanisms at both levels of control. For instance, when the logically centralized global controller is unavailable, B4 will replicate the global controller across multiple wide-area networks. When a local site-level controller is unavailable, a new controller will be appointed from the set of reachable standby controllers.

Similar to B4, Espresso also has a hierarchically physically distributed architecture, which has higher reliability and full interoperability with the internet and heterogeneous peers [325]. Espresso has a fail-safe static system where the data plane keeps the last known good state to allow for control plane unavailability.

A recursive building block known as a logical xBar, which is a programmable entity that can switch packets between ports, and a logical server, which manages forwarding tables and control plane computations, are used to create a centralized hierarchical control plane [326].

Logically Distributed and Physically Distributed Control

The main feature of conventional SDN is the logically centralized control plane, which decouples the control and data planes and yields better management of intra-domain networks. However, a logically centralized control plane has caused poor performance in inter-domain networks, such as controlling heterogeneous networks, so a logically distributed control plane architecture has been proposed for better control of such networks [327]. In a logically distributed control architecture, control is distributed among several controllers. The logically centralized control architecture is not preserved in the logically distributed control architecture, as shown in Figure 7d. As evident from Figure 7d, there is no communication between a centralized controller and forwarding elements, as there is no centralized SDN controller in this control architecture. However, forwarding elements in each network portion involve communication with a local controller for obtaining network control. In this architecture, a given controller does not broadcast all its network modifications to other controllers, unlike in the logically centralized flat architecture. Modifications in a given network will be transferred to other controllers only when needed (such as in the case of inter-domain service), where controllers will explicitly communicate in such an instance.

A Distributed SDN Controller (DISCO) is a logically distributed and physically distributed controller framework implemented on FloodLight that is proposed for modern heterogeneous networks in a multi-domain environment, where a given DISCO controller will manage its own network, communicate with other controllers using an inter-domain part to provide network services and share aggregated network-wide information (reservation, topology state modifications, disruption), and use an intra-domain part for the main functions of the controller such as network monitoring, reacting to network issues, etc. [328]. DISCO can adapt to heterogeneous multi-domain network topologies to make sure that link failures are mitigated and end points are migrated.

In D-SDN, the controller is distributed both logically and physically, which achieves logical decentralization by using a hierarchy of controllers where main controllers delegate control to secondary controllers to manage certain devices [309]. D-SDN has enhanced fault tolerance and security compared to DISCO.

Software Defined Exchange (SDX)-based controllers are used to connect participants of different domains using a shared platform, enhancing control over inter-domain traffic management to improve wide-area traffic delivery. SDX, which is enhanced by logical and physical distribution of control, relies on inter-domain control policies to provide end-to-end services such as load balancing, traffic redirection, etc. However, SDX-based controllers have shown security vulnerabilities and reliability issues [329]. Several variations of

SDX based controllers have arisen, such as Cardigan [330], AtlanticWave SDX [331], etc. Cardigan is a distributed SDX controller that applies routing as a service abstraction to a RouteFlow-controlled IP network in order to reduce operational complexity. The AtlanticWave SDX controller is able to provide multiple paths dynamically on demand, apply QoS, prioritize policies, manipulate flow levels, etc. in a distributed manner.

Hybrid Control

A hybrid control architecture implements a combination of fully centralized and fully distributed architectures. In other words, both centralized and fully distributed versions of the control plane exist in a hybrid architecture. Based on the network conditions, the hybrid architecture has the flexibility to adjust the distributed or centralized features from zero to full, as shown in Figure 7e. As seen in Figure 7e, control plane communication also involves receiving control messages from the centralized controller and exchanging control messages between the nodes. In this architecture, the data plane elements can be involved in decision-making and network control. This architecture is called hybrid, as both the data plane and the control plane are involved in network control. This architecture has higher scalability and higher resilience to failures due to the hybrid mode of operation. For instance, when communication with the centralized controller is unavailable, this architecture can run in full distributed mode, and vice versa, depending on the network conditions. As an example of the hybrid mode, since the SDN controller has an overview of the network, instead of sending whole flow tables to the nodes, it can instruct the nodes to use a specific routing algorithm, but nodes should exchange control messages in a distributed manner to build the flow tables.

Dey et al. have proposed hybrid routing to address scalability issues found in the logically centralized architecture, where a local router in each data plane element handles some part of routing and most of routing is controlled by the controller [332].

Considering the coexistence of legacy switches and SDN switches, consistent forwarding graphs are constructed for coordination of forwarding of SDN switches and distributed routing in hybrid control architecture, which also secures a high throughput while maintaining forwarding consistency [333].

DevoFlow (Devolved Flow) is a hybrid control approach that reduces frequent interactions with the control plane and the data plane in order to reduce overhead and delay by distributing some of the control plane to the data plane. In the DevoFlow approach, the controller is only involved in controlling significant and long-lived flows, whereas switches make local control decisions [310].

Distributed Flow Architecture for Network Enterprise (DIFANE) is a scalable hybrid control architecture where authority switches are assigned rules by the controller, which has an algorithm to partition rules and minimize rule fragmentation [334]. DIFANE keeps all traffic in the data plane by selectively directing packets through intermediate switches that store rules.

OpenRouteFlow, which enables legacy routing as a software-defined routing service that provides path-oriented and traffic-oriented subscription and publication services for different scenarios of network control, has been used to achieve hybrid network control [335].

Fibbing is a hybrid control framework where the controller tricks the routers into seeing a fake topology that is constructed to achieve the desired forwarding information base. It applies central control over traditional distributed link state protocols such as Open Shortest Path First (OSPF), where computation and installation of forwarding information base on the data plane are carried out by traditional distributed protocols [336].

Hybrid control is secured in HybridFlow, which is a control architecture that controls legacy devices using an SDN control plane that abstracts the hybrid SDN network as a logical SDN network. HybridFlow, which was originally implemented in the POX [337] controller, maps logical ports of the SDN network to physical ports of the actual network [338].

Table 6 summarizes the classification and details of controller frameworks based on the degree of distribution of control.

Table 6. Summary of details of different controller frameworks.

Architecture	Name	Northbound IF	Developer	Found Year	Open-Source	Consistency	Scalability	Progr. Lang.
Logically and physically centralized control	NOX [273]	Ad Hoc-API	Nicira networks	2008	Yes	Strong	Low	C++
	NOX–MT [312]	Ad Hoc-API	Nicira networks	2012	Yes	Strong	Low	C++
	POX [337]	Ad Hoc-API	Nicira networks	2013	Yes	Strong	Low	Python
	Beacon [313]	Ad Hoc-API	Stanford University	2013	Yes	Strong	Low	Java
	Floodlight [278]	RESTful-API	Big-switch networks	2012	Yes	Strong	Low	Java
	SE-Floodlight [314]	RESTful-API	Big-switch networks	2013	Yes	Strong	Low	Java
	Trema [315]	Ad Hoc-API	NEC corporation	2012	Yes	Strong	Low	C, Ruby
	Ryu [316]	Ad Hoc-API	NTT labs	2013	Yes	Strong	Low	Python
	Meridian [317]	Extensible-API	ONF	2013	Yes	Strong	Low	Java
	Rosemary [275]	Ad Hoc-API	ON Lab	2014	Yes	Strong	Low	C
	Maestro [318]	Ad Hoc-API	Rice University	2011	Yes	Strong	Low	Java
	ParaFlow [274]	Abstract-API	Beihang University	2017	Yes	Strong	Low	C++
Logically centralized, physically distributed with flat control	SMaRtLight [319]	RESTful-API	University of Lisbon	2014	No	Strong	High	Java
	HyperFlow [298]	N/S	University of Toronto	2010	No	Weak	High	C++
	ONOS [280]	Intent, REST API	Open Networking lab	2014	Yes	Weak, Strong	High	Java
	OpenDaylight [277]	Intent, REST API	Linux foundation	2013	Yes	Weak, Strong	High	Java
	Onix [300]	NVP NBAPI	Nicira Networks	2010	Yes	Weak, Strong	High	Python, C
	Ravana [320]	N/A	Princeton University	2015	No	Strong	Low	Python
Logically centralized, physically distributed with Hierarchical control	Kandoo [308]	Java RPC	University of Toronto	2012	Yes	Strong	High	C, C++, Python
	Orion [322]	N/A	Tsinghua University	2014	Yes	Weak	High	Java
	B4 [324]	RESTful-API	Google	2013	No	Strong	High	Python, C
	Espresso [325]	RESTful-API	Google	2017	No	Strong	High	Python, C

Table 6. *Cont.*

Architecture	Name	Northbound IF	Developer	Found Year	Open-Source	Consistency	Scalability	Progr. Lang.
Logically and physically distributed control	DISCO [328]	RESTful-API	Thales communication and security	2014	No	Strong	High	Java
	SDX [329]	RESTful-API	Princeton University	2014	Yes	Strong	High	Python
	Cardigan [330]	RESTful-API	Victoria university of Wellington	2013	Yes	Strong	High	Java
	AtlanticWave [331]	RESTful-API	FIU, GIT	2015	No	Strong	High	Python
Hybrid control	DevoFlow [310]	RESTful-API	University of Waterloo	2011	No	Strong	High	Java
	DIFANE [334]	RESTful-API	Princeton University, AT&T labs	2010	Yes	Strong	High	Python
	OpenRouteFlow [335]	RESTful-API	Tsinghua University	2015	Yes	Strong	High	Python
	Fibbing [336]	N/A	Princeton University	2014	No	Strong	High	Python, C
	HybridFlow [338]	RESTful-API	Fudan University	2016	Yes	Strong	High	Java

5.3.2. Packet vs. Flow Control

In conventional networks, the basic unit of control is the packet, where packets are routed using the information contained in the headers, such as source IP, destination IP, etc. Thus, the highest level of granularity of control is packet control. Fine-grained packet forwarding refers to the ability to direct traffic using specific criteria, such as packet content.

The following are some advantages of fine-grained packet control. Due to its high level of granularity, fine-grained control of traffic can increase flexibility and decrease network congestion. Additionally, it enables the development of network rules tailored specifically for each application to enhance performance. Additionally, it can boost network visibility, which helps with debugging.

The following are some disadvantages of fine-grained packet control. It may introduce more overhead in control plane communication as individual rules for each packet need to be installed in the switches by the control plane, thus increasing the latency and slowing down the network. It can be complex to implement in large networks, such that its scalability is low. An attacker may use fabricated fine-grained forwarding to forward traffic to a compromised server by bypassing security measures.

The traditional OpenFlow protocol implements packet-based flow rules that send a packet-in message to the controller when the switch is unaware of the flow rule for a packet (when a flow table mismatch occurs), where the controller will compute routes and update the flow table entry of the switch. This reactive approach of OpenFlow for packet control is a high-granularity control approach that introduces vulnerabilities in denial of service attacks as well [339]. A routing protocol for cluster-based KDN where packet matching for flow tables using 14 header match fields has been proposed in research [340]. A fine-grained traffic monitoring system that generates flow rules for such fine-grained traffic using a Markov decision process and a double-deep Q-network algorithm constrained by the expected level of statistics details and flow-table limits has been implemented as a high-granular control method [341]. A technique for reducing overhead for control plane communication for packet forwarding, where a buffer is created in the switch for flow table mismatches and collected mismatched entries are buffered and sent to the controller to reduce control plane overhead and increase switch overhead, has been studied in [342]. A system that attempts to prevent IP spoofing attacks using an IP source validation scheme that has granularity in IP packets, with subnet-level granularity for intra-domain and IP address-based granularity for inter-domain, has been studied in [343]. Considering the fact that the controller enables a fine level of granularity for controlling traffic and creating forwarding rules, the work in [344] shows that attackers can predict and reconstruct these forwarding rules. A unified controller known as Magneto for fine-grained path control in KDN and legacy hybrid networks, which uses magnet MAC addresses to dynamically map IP addresses to magnetic MAC addresses at hosts using the address resolution protocol, has been used for routing [345]. A fine-grained security policy implementation has been feasible for the controller with the aid of virtual networks, which deny unauthenticated access to the network [346]. A fine granular inter-domain routing matching multiple IP header fields with the aid of the OpenFlow protocol for routing control can also reduce redundant flow entries for inter-domain settings [347].

When thinking in terms of applications, they send packets as a flow of many packets, such as QoS-based flows. Thus, applications may be better served using flow-based control. Furthermore, abstraction of flows can result in aggregated flows that can be matched to obtain the required task. Let us briefly discuss the pros and cons of coarse-grained flow control.

Some advantages of coarse-grained flow control follow. In flow-based decision making, the overhead for control plane communication is low. In most cases, only one packet of the flow may be required to be inspected, thus reducing the complexity of the overall system. The network is easier to manage and control, as scalability is high with this approach.

Furthermore, security is high, as traffic from compromised sources can be quickly identified and isolated.

The following are some disadvantages of coarse-grained flow control. This approach is not as precise as fine-grained packet control and is less flexible, as it limits the ability of the network to adapt to changing traffic patterns or respond to unexpected events. Furthermore, it has less visibility of the network, as individual packets within a flow are not monitored, which can increase difficulty in troubleshooting problems.

Admission control with flow aggregation for QoS provisioning, which uses a model to analyze the required amount of bandwidth and buffer space at OpenFlow switches to meet requirements in delay and packet loss, has been utilized [348]. A local fast rerouting technique in case of link failures that aggregates traffic flows affected by the failure to compute a local reroute path by the controller for the aggregated flow that reduces the number of flow operations between the controller and the switch has been studied in [349]. A flow control mechanism that manages QoS and best-effort flows based on network slices and schedules, which uses multi-objective optimization for priority forwarding, multipath forwarding for best-effort flows, and an algorithm for flow allocation and slice adjustment, is presented in [350]. The controller can utilize traffic flows for intrusion detection by gathering statistical information about the flows from OpenFlow switches and analyzing the flows by aggregating a set of features [351]. A framework known as software-defined label switching combined central control with label switching to reduce storage burden while maintaining per-flow control, is a hybrid approach that incorporates part of the control plane into switches and has resulted in a lower number of flow entries and overflows [352]. A framework for minimizing the number of flows while maintaining end-to-end delay as a QoS parameter that aggregates flows using an algorithm for flow control has been studied in [353]. A framework to mitigate the denial of service attack on the controller by effectively rerouting malicious traffic, adjusting flow timeouts, and aggregating flow rules has been achieved in [354]. Considering the fact that fine-grained traffic flow measurement is challenging due to a lack of monitoring resource constraints, a traffic aggregation and measurement paradigm that aggregates flows and uses them to estimate network flows using optimization under the constraint of flow table size has been studied in [355].

5.3.3. Reactive vs. Proactive Control

In the reactive control mechanism, network changes are triggered based on events or traffic patterns. Reactive control mode involves switches contacting the controller for decision-making, such as when the flow table does not have any entry for the flow/packet (arrival of a new flow), so the controller has to be consulted, where the controller will compute the path based on its policies and send the instructions for configuring the path for the flow table as a reaction to the request by the switches.

Advantages of reactive control include that, as the configuration of the switches occurs only when needed, it will result in efficient use of network resources and reduced communication overhead. Reactive control supports dynamic network control such as adding or removing switches without requiring a network-wide configuration, and it can also adapt to changing traffic patterns in real time.

When considering its disadvantages, one of the main concerns with reactive control is that it can provide a base for attackers to use trigger events to manipulate network behavior or cause denial of service attacks. Furthermore, it can act as a hindrance for the controller to obtain a global view of the network, as the controller only computes and installs flow rules in response to trigger events. In addition, it can also cause unnecessary delays for time-critical applications, as a trigger event is required for the controller to take an action.

The Procera language has been introduced as a language that supports reactive control and includes a declarative policy language based on functional reactive programming [285]. A reactive approach for mitigating reconnaissance attacks using shadow servers is presented in [356]. A study that attempts to quantify the number of control plane messages

exchanged between an ONOS controller and data plane elements has found that reactive control results in a lower number of message exchanges, such that it addresses the scalability issue of KDN [357]. A study that analyzes the traffic traces involved in the reactive flow table update mechanism of OpenFlow states that there is a trade-off between the size of the flow table and the rate of installation of a missing rule [358]. A security framework with a reactive approach that analyzes potential attacks and isolates attackers for an industrial network has been studied in [359]. A reactive, stateful firewall for KDN that filters TCP communication according to network policies and processes the traffic into a set of OpenFlow rules with the help of a finite state machine of protocols has been studied [360]. A reactive control approach for mitigating security attacks such as port scanning, flooding, ARP spoofing, etc. using exception control packets has been studied in [361]. The study in [362] highlights that DDoS attacks can occur due to the reactive approach to flow rule installation in KDN and analyzes a set of strategies to overcome the vulnerability.

In proactive control, the network controller pre-programs the forwarding elements with a set of rules to handle all possible traffic flows before the traffic arrives at the switches. In this approach, traffic is analyzed using different techniques, such as machine learning or algorithms, and control is provided before problems occur.

Advantages of proactive control include that network downtime can be reduced by taking proactive measures to maintain network availability. Security can be improved by proactively implementing security policies before threats enter and spread through the network. In a proactive approach, the overall quality of service can improve, such as with less latency and high throughput, as network switches are programmed in advance.

Disadvantages of proactive control include that it has limited flexibility, as it does not dynamically adjust to changes in network traffic. It also results in a high communication overhead, and the system becomes complex when the network is large, thus having low scalability. Proactive control has difficulty adapting to new or unknown threats, as precomputed policies may not affect such threats.

A proactive approach for mitigating reconnaissance attacks using the technique of IP and port shuffling is presented in [356]. Network congestion control using a joint reactive and proactive approach that couples the activity of both users and the network controller using an optimization technique has been studied in [363]. In proactive flow management, all required flow entries are installed in the switches by the controller, which causes additional requirements such as switch memory hierarchy optimizations [364]. A proactive approach to protecting blockchain nodes from domain name service amplification attacks by using a stateful mapping scheme, an entropy calculation scheme, and a DDoS mitigation module has been effective [365]. A proactive attack and failure resilience mechanism that uses a sandboxing technique to isolate the controller from the host and employs live remote checkpointing and migrating between different hosts to evade failures and attacks has been studied in [366]. A framework that can proactively adapt to the attack surface of networks and dynamically optimize defense strategies using moving target defense techniques has been realized in a cost effective manner [367]. A proactive admission control and resource management strategy for virtualized networks that maximizes the high-priority networks subjected to substrate limitations and memory requirements of virtual network requests is presented in [368]. A proactive routing rule placement technique for KDN that updates the flow rules proactively, however, considering whether the routing path actually changed or not, has been studied in [369]. By enabling the controller to add backup paths proactively along with the working paths and enabling switches to perform recovery actions locally, a failure recovery mechanism for KDN has been studied in [370].

5.3.4. Fully Consistent vs. Eventually Consistent Control

Data consistency is an essential factor in distributed control KDN architectures, specifically in the logically centralized and physically distributed flat control architecture. A controller's consistency has three aspects namely: state consistency, rule-update consistency, and version update consistency [371]. Version update consistency guarantees that

version updates between the controllers are consistent. State consistency ensures that all controllers have the same global view. Rule update consistency ensures that switches under a controller have the same forwarding policies.

Full/Strong consistency ensures that all the controllers have the same view of the data at all times, and any update made on the network is immediately visible to all controllers, which guarantees that the data will remain consistent at all times. However, full consistency can lead to high latency and reduced scalability.

The eventual/weak consistency allows a delay between the updates in the network and the time at which those updates are reflected in all the controllers. Thus, for a short period of time, there can be different views by different controllers. Eventual consistency has higher scalability compared to full consistency.

In dynamic control, when a set of switching devices is handed over from one controller to another during load balancing or in the event of a controller failure, the forwarding rules in the switches may need to be updated [372]. Distributed consensus-based strong consistency algorithms such as Raft have shown poor performance under controller overload conditions and poor scalability and responsiveness despite their strong consistency [373]. On the other hand, eventual consistency models, even though they have high scalability, availability, and communication overhead, do not have tolerable consistency, as there is a time period during which the updates are not consistent [374]. As both methods have pros and cons, some have adapted state consistency using a hybrid approach, having both full consistency for critical operations that affect a large portion of the network and eventual consistency for less critical changes [375].

Onix and ONOS are examples of controllers that have both full and weak consistency levels. On the other hand, the SMaRtLight and Ravana controllers have strong consistency, while the Hyperflow controller has weak consistency [376].

5.4. Controller Functions

5.4.1. Data Collection

In KDN architecture, the control plane collects data to aid in decision-making in the control plane (either using an information-centric approach using direct data/information or using a knowledge-centric approach by generating knowledge and rules from the collected data), whereas management-related data (data for network monitoring and configuration) and data specifically for knowledge generation are collected by the management plane. Note that most of the data are collected by the management plane, while only specific data that is required to aid control decision-making is collected by the controller in the KDN architecture. However, some data, such as data related to network topology and network traffic flow, will be jointly collected by the control and management planes in order to develop a global view for making management and control decisions.

The control plane collects detailed statistics about the network traffic, such as the number of packets and bytes transmitted, the rate of traffic flows, and the types of protocols used. By collecting detailed traffic statistics, greater visibility into the behavior of the network can be obtained, which will allow early detection of issues before they become critical. By analyzing network statistics, the control plane can identify congested links and adjust traffic flows to balance network utilization and improve performance. Network statistics can be utilized to be confident that network traffic is prioritized according to QoS demands [377].

The control plane can collect QoS data such as latency, throughput, packet loss rate, bandwidth utilization, etc. in order to make sure that network traffic is prioritized and delivered adhering to user requirements [378].

The control plane may collect data regarding policies of the data plane elements, such as access control policies, QoS policies, routing policies, and security policies, in order to verify policies [262].

The control plane needs to collect data on security events such as network attacks and intrusion attempts. These data will be used to identify potential security threats and trigger policies that can mitigate the impact of these events, as the control plane is responsible for enforcing security policies and responding to security events in real time. Furthermore, it can provide security-related data to the knowledge plane to generate knowledge and rules to refine application policies related to security [379].

The control plane may need to collect information about routing protocols, including the configuration and behavior of the network devices, which will be necessary for traffic engineering using the controller [380].

5.4.2. Path Computation

Path computation involves determining the optimal path that data packets should take through the network based on various factors such as network topology, traffic flow, and network policies. As the control plane has a global network view, that information can be used to compute the most efficient path that a data packet can take to reach its destination using an algorithm (such as shortest path algorithms) [381]. However, some paths may be congested. Thus, making a decision only using the topology may not be ideal. Therefore, the controller can consider the network traffic flow to identify congested links in order to avoid them when computing the paths [382]. Furthermore, when routing, the controller can enforce network policies such as encrypting sensitive data and forwarding it through secure channels or giving priority to one type of traffic over others [383]. Instead of proactively computing paths, paths may be computed on demand, such as due to changes in a network topology or traffic flow. For instance, if a link fails or becomes congested, the control plane can compute paths for the affected data packets reactively [384].

5.4.3. Policy Enforcement

Note that policy enforcement can occur on both the management and control planes. However, each of these planes will enforce different policies that are identical to each plane. As discussed in Section 4.5.2, the management plane is responsible for enforcing configuration and monitoring policies. As discussed in Section 5.4.1, the control plane is responsible for enforcing access control policies, QoS policies, routing policies, and security policies. The policy flow in a KDN environment [247,385] is graphically illustrated in Figure 8.

Before policy enforcement by the control plane, first, policies must be defined by the network administrators, which specify how network devices must behave. These policies, which are applicable to the control plane, can cover a wide range of applications such as routing, security, access control, quality of service, etc. [386]. The control plane can enforce/translate policies by programming network devices to perform specific actions once certain conditions are met using a policy engine. A policy engine is used to translate the network policies into rules by considering other data, rules, and knowledge [385]. As evident from Figure 8, the controller policy enforcement and verification module generates flow rules by comparing the collected data with other data such as network topology and device capabilities, knowledge/rules from the knowledge plane, policy verification output from the policy verification module, and the requirements of the policy. As an example of policy enforcement, it can program switches to drop traffic that does not confirm to QoS policy, or it can program firewalls to block traffic that violates security policies [387]. The controller can use protocols such as OpenFlow to program policies (in the form of flow rules) into forwarding elements. From time to time, the control plane will verify that policies enforced are correctly functioning in the data plane. For achieving that purpose, the controller will collect policy data from the data plane and compare it with the existing policies, as shown in the policy verification module in Figure 8. The policy verification module will output the nodes' identifying information that does not adhere to the existing policies. If any policy violations are detected, the controller can take corrective action by sending the flow rules generated by the policy engine for such nodes [388]. A dynamic

policy update in response to a change in a network condition, such as updating security policy upon detection of an intrusion in the network to block the threat, can be used to achieve the automation desired in the KDN paradigm, where network policies are dynamically updated to adapt to changing network environments with minimum human intervention. Policy updates can occur in the application plane based on knowledge/rules output from the knowledge plane in response to real-time network activity, as shown in Figure 8.

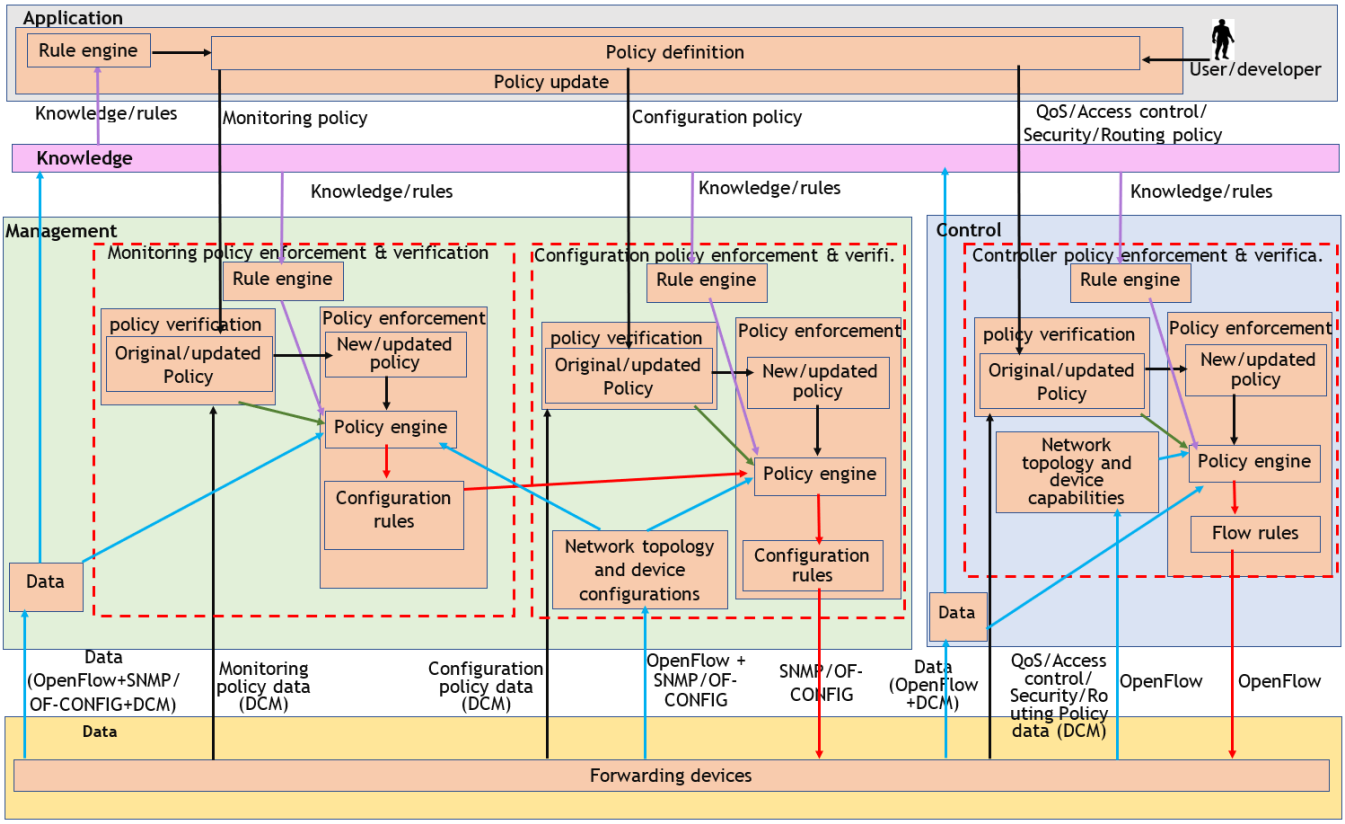

Figure 8. Network policy life-cycle in a KDN.

Note that Figure 8 also shows the policy lifecycle for network monitoring and network configuration in the management plane. As evident from Figure 8, in the monitoring policy verification and enforcement module, configuration rules are generated from the policy engine based on network topology and configuration, monitoring policy verification output, data and knowledge, and network monitoring policy [247], as discussed in Section 4.2. The generated configuration rules from the monitoring policy enforcement and verification module will be compared with other data, knowledge, configuration policy verification output, and configuration policies to update the configuration rules and be provided to the data plane [389], as evident from the configuration policy enforcement and verification module in Figure 8.

5.4.4. Traffic Engineering

Traffic engineering involves optimizing the flow of traffic to improve network performance and reliability. The control plane can monitor traffic in real-time and detect areas of the network that become congested. The control plane can distribute network traffic evenly across multiple paths in the network, which will help avoid congestion on specific links and improve performance. In [390], according to bandwidth utilization, traffic is redirected from congested links by the centralized controller to enhance the efficiency of link usage in terms of throughput, jitter, and packet loss rate. The controller can also allocate resources,

such as bandwidth, in affected areas of congestion to reduce the congestion [391]. Furthermore, the control plane can enforce QoS policies to prioritize certain traffic over others. For instance, video traffic can be given higher priority than file transfer traffic, as video traffic is time-critical, in order to increase the overall quality of experience of the user [392]. The control plane should ensure that traffic is rerouted in the event of a network failure. It can use protocols such as Multi Protocol Label Switching (MPLS) for fast and reliable rerouting of traffic, which will help minimize downtime and ensure network reliability. Specifically, a protection scheme using pre-computed backup paths by the controller to use MPLS routing, which guarantees instantaneous recovery time and a zero packet loss rate after failure detection, can be employed [393]. Finally, dynamic policies can be used to adjust routing policies to improve the efficiency of network traffic in response to network change events. For instance, if the controller detects that network devices' resource utilization is high, it can enforce a policy to use a routing protocol that uses less device resources [394].

5.4.5. Network Virtualization

There are network virtualization technologies such as Virtual Local Area Networks (VLANs), Virtual Private Networks (VPNs), etc. The control plane can build virtual networks on top of real networks by utilizing these network virtualization technologies. Network virtualization will increase network scalability and flexibility, since they can be generated dynamically upon requirement [395]. Furthermore, these virtual networks can be separated from one another to stop unwanted access to sensitive data and ensure high security in the network. Service providers could offer services to several customers on a single physical network thanks to these virtual networks, each of which may have its own set of users and resources [395]. In addition to virtual networks, the control plane can leverage Network Function Virtualization (NFV) to virtualize network services that can be installed as needed, including firewalls, load balancers, routers, etc. [396]. NFV successfully lowers service providers' capital and operating costs.

5.4.6. Service Chaining

Service chaining involves directing network traffic through a sequence of network services such as intrusion detection systems, load balancing, firewalls, etc. [397]. In policy-based service chaining in conventional SDN, the sequence of network services is predefined based on network policies, such as those enforced by the network administrators. In dynamic service chaining, the network functions in the service chain can be updated based on varying network conditions, such as real-time network traffic, which can be used in KDN architecture. For example, in a KDN, by anomaly detection using the knowledge plane, it can be detected whether there is a threat to the network or not. If a threat is detected, the service chain can be updated by adding a firewall [398].

6. Data Plane

6.1. Introduction to Data Plane

The bottom layer of the KDN architecture is the data/infrastructure plane, consisting of forwarding elements that are interconnected using transmission media. This layer's architecture and performance in KDN are very similar to those of SDN. However, there are a few differences in the data plane of KDN compared to SDN. First, the data plane is managed (configured and monitored) by the management plane and controlled (flow rule installation containing forwarding rules, installing access control rules, prioritizing network traffic based on QoS, implementing security policies, etc.) by the control plane. Furthermore, in KDN, more data are collected by management and control planes compared to SDN. Extra data collected in KDN are used for knowledge generation, as knowledge is used in making network control decisions by the controller, network monitoring and configuration decisions by the management plane, and to dynamically update application plane policies. Due to the extra communication burden utilized for sending data to management and

control planes, forwarding elements in KDN require more resources, such as bandwidth and processing power, compared to switches in SDN.

Let us discuss the components of the data plane in detail, as given in the following subsections.

6.2. Forwarding Devices

6.2.1. Architecture of a Forwarding Device

The architecture of a KDN forwarding device, along with flows between management and control planes [399], is given in Figure 9.

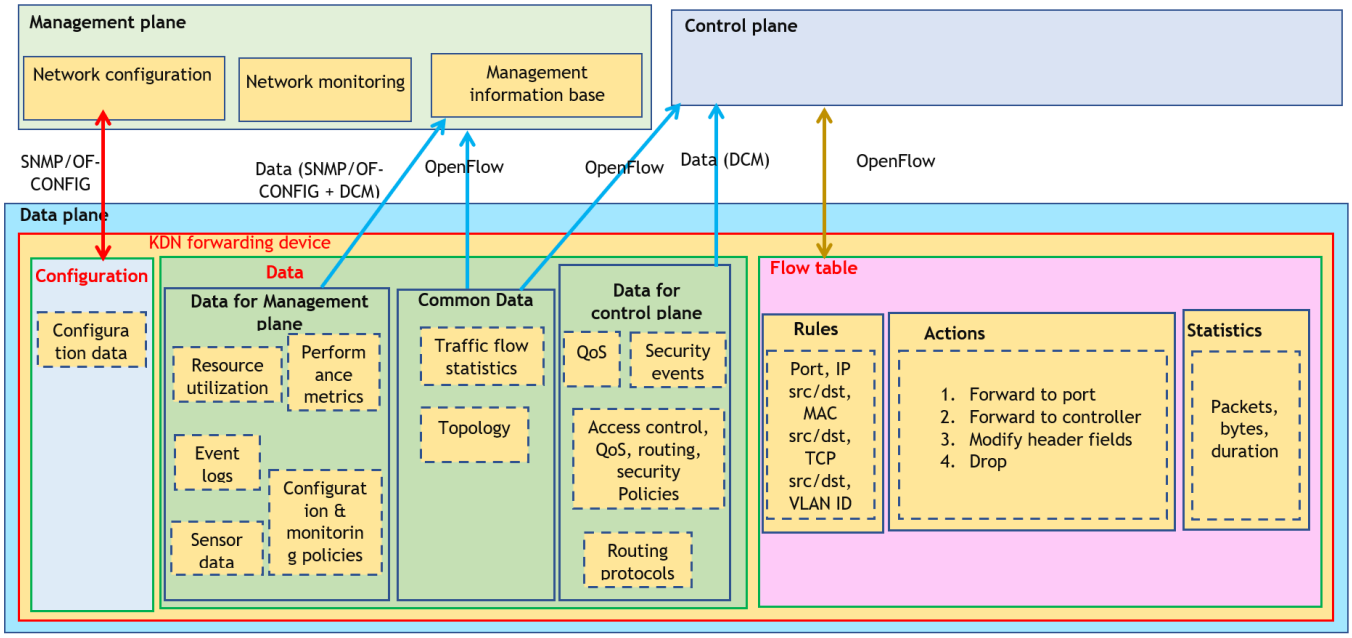

Figure 9. KDN forwarding device architecture in data plane with different flows exchanged with other planes.

The forwarding devices can be physical/virtual switches, routers, wireless access points, etc. These devices are configured and monitored by the management plane, while flow rules are installed by the controller, as evident from Figure 9. Note that, compared to an SDN switch, a KDN switch has data to be sent to the management and control planes. Note that, as evident from Figure 9, device topology data and traffic flow statistics data are jointly collected by the management and control planes (using OpenFlow), while data such as resource utilization, event logs, sensor data, etc. are collected by the management plane (using SNMP/OF-CONFIG + DCM), while data such as QoS data, security events, routing protocols, etc. are exclusively collected by the controller (using a DCM). Forwarding devices do not have the intelligence for control in order to take autonomous decisions, as all rules are explicitly defined by a logically centralized controller. As discussed previously in Sections 4 and 5, and as evident from Figure 9, protocols such as SNMP/OF-CONFIG can be directly used for device configuration and to collect configuration data, while protocols such as OpenFlow can be used for flow rule installation.

Figure 9 depicts a pure KDN switch with data flows, where only KDN-based forwarding using flow tables takes place. However, the pure KDN switch architecture is difficult to integrate with legacy networks. In Section 5.3.1, we presented the high-level architecture of a hybrid KDN switch. Note that, in the presented hybrid switch, the KDN module refers to the pure KDN switch presented in detail in Figure 9. Note that the local control agent is not implemented in a pure KDN switch, while the control channel is directly connected to the KDN module in that switch.

As shown in Figure 9, an OpenFlow flow table basically consists of three fields called rules, instructions, and statistics. Rules contain the switch port, source MAC address, destination MAC address, source IP address, destination IP address, source TCP address, destination TCP address, Virtual Local Area Network (VLAN) ID, etc. to match against incoming packets. Once matched, one of four actions—forward to port, forward to controller, modify header fields, or drop packet—will be taken. If there is a mismatch, the corresponding action for the incompatible (unmatched) flow table entry will be carried out. The action for a mismatch can be one of dropping the packet, searching in the next flow table, or forwarding to the controller. In order to request the controller in case of a missing flow table entry, the forwarding device should send a "packet-in" message to the controller, where the controller will compute the paths and send a "Flow-mod" packet back to the switch to modify the flow table with forwarding information for the packet. Furthermore, the controller can collect flow statistics exclusively maintained by the flow table in the statistics field using "Switch Feature" messages and event-based messages such as port changes using "CPort-status" messages.

6.2.2. Physical Switches

Switch Processing

Compared to legacy network forwarding, which is IP or MAC address-based, OpenFlow switching can occur based on TCP addresses, VLAN IDs, or switch ports, which increases the length of the rule field and increases the processing complexity of the flow table. Thus, recent work has proposed integrating a Graphic Processing Unit (GPU) along with a CPU to accelerate packet processing in switches by utilizing the GPU's parallel processing capability [400]. A Network Processing Unit (NPU) from vendors such as Intel, Cavium, Broadcom, etc. has been utilized as the processing unit in switches [401]. Some have proposed network processor-based acceleration cards to improve the processing capability of the switches [402]. Some have proposed a hardware implementation of the OpenFlow programmable packet parser using a Field Programmable Gate Array (FPGA) to increase the switch speed and reduce the waiting and service times [403]. Recent research investigates the feasibility of a traffic offload hybrid switch using an Application Specific Integrated Circuit (ASIC) and FPGA. This switch proposes to offload switching between two different technologies: fast ASIC and programmable FPGA based on volume, priority, volatility, etc., in order to yield high throughput and low latency [404]. Some researchers have proposed using a combination of CPU and ASIC for processing different types of flows. In particular, to handle frequently occurring small flows, a CPU with relatively lower processing power is proposed, while to handle large and non-frequent traffic, an ASIC implementation of the switch is proposed [405].

Switch Memory

In order to be compatible with legacy networks, most modern switches are manufactured as hybrid switches, which support traditional forwarding based on routing protocols in the switch combined with logically centralized control-based forwarding in KDN [406]. However, pure KDN switches only support KDN and do not support traditional routing protocol-based switching. Ternary Content Addressable Memory (TCAM) is the most widely used memory for forwarding elements in KDN, as it provides high-speed lookup and forwarding capabilities, making it well suited for implementing the match-action tables used in switches. However, TCAM is expensive and less power efficient, and the TCAM capacities of current OpenFlow switches on the market can vary in a vast range from a few thousand to a few millions [407]. Other memory technologies, such as Dynamic Random State Memory (DRAM) or Static Random Access Memory (SRAM), do not provide the same level of speed and efficiency as TCAM [408]. However, SRAM is more flexible and scalable. Thus, there are proposals to use a combination of TCAM and SRAM in order to have high flexibility and high packet classification performance [409].

Flow Rule Management

A forwarding device is responsible for packet forwarding based on the flow rules installed in its memory, with the help of a local processor. Thus, the main function of a KDN switch is to process packets (forward based on installed flow rules) and collect and send data to management and control planes. However, a given forwarding device has limited memory and processing power. When the size of the network becomes large, the number of flow rules in a given switch can also increase, so efficient flow rule management techniques can be used to manage flow rules with limited space. Such management techniques should make sure that an optimal number of flow rules are maintained without removing flow rules that could result in violating network policies and constraints. Filling up the switch memory can result in packet dropping and provoke frequent communication with the controller to search for unknown rules, thus degrading the controller's performance. OpenFlow rules are more complex and consume more memory than forwarding rules in conventional IP-based routers. Assigning a timeout to different flows in the flow table such that, once the timeout is reached, such flows will be automatically removed from the flow table is one way to manage flow table entries [410]. Another approach to managing the flow table is to aggregate flow rules using prefixes or by splitting and distributing flow rules [411]. Furthermore, flow rule caching using a two-tier approach has shown better switch memory utilization [412]. Some have used OpenFlow switches to handle short flows using flow rules, while large flows are handled by contacting the controller, which has resulted in a significant reduction in flow-table entries [310]. Some researchers have attempted to use distribution frameworks to decompose large flow tables into smaller ones and optimally distribute them across the network while preserving the policies [413].

Forwarding Models

As reviewed in Section 5.2.2, there are numerous proposals for the implementation of switch forwarding models using protocols such as OpenFlow [291], ForCES [292], OpFlex [293], POF [294], PCE–PCC [295], OpenState [296], etc. OpenFlow and ForCES both have a completely logically separated control plane from the data plane, whereas in OpenFlow, the control plane is typically physically separated from the forwarding devices, while in ForCES, the control plane can exist in the same device. In OpFlex, a part of the control plane is redistributed to forwarding devices in order to improve the scalability of KDN, even though the policies are logically centralized. Compared to OpenFlow, in POF, flow instruction sets are used for packet forwarding, where flow table matching using header fields is not necessary, which speeds up forwarding in switches. In OpenState, OpenFlow match-action is extended to finite state machines for doing stateful tasks inside the switches, which can be considered again as an attempt to decentralize some control back to forwarding elements. However, OpenFlow is the most widely used implementation for switches in KDN, as it implements the original SDN concept of complete decoupling of the control plane from forwarding elements.

Table 7 summarizes existing protocol details that implement switch forwarding models.

Hardware Implementation

Switches can be implemented in general PC hardware, open network hardware, or vendor hardware. In general, in PC hardware implementations, the forwarding device is implemented as software running on the host operating system [414]. A fast data path based on caching of flow table entries in onboard classification hardware on the network interface card to improve lookup performance of PC-based OpenFlow switching in Linux has been presented in [415]. The authors in [416] highlight that the performance of PC-based switches is lower compared to dedicated hardware-based switches due to the fact that packet input–output in the operating system's network stack is a significant bottleneck. This work analyzes several PC-based switch implementations such as Linux IP forwarding, Linux bridge, Open vSwitch, Data Driven Packet Processing (DDPK), and Layer 2 Forwarder (L2FWD) and shows that the performance of the DDPK is nearly six times that of the Open vSwitch. Open network hardware-based switches are mostly used for testing and development of forwarding devices, which provides a vendor-independent

platform to build switches. NetFPGA is an open network hardware-based platform to build high-performance networking systems in hardware, which uses FPGAs to implement core data processing functions [417]. Using NetFPGA as the platform, SwitchBlade and ServerSwitch are practical examples of the implementation of physical switches using open network hardware [418]. Finally, vendor-specific switches provide several advantages over open network hardware-based switches, such as better integration with products from the same vendor, ease of deployment, support for specialized features from the vendor, etc. However, there are some drawbacks, such as vendor lock-in, reduced interoperability with other vendors, limitations in flexibility and scalability, etc. Cisco, Indigo, IBM, and Juniper Networks are a few examples of vendors that create vendor-specific switches in KDN [419].

Table 7. Summary of protocol details that implement different switch forwarding models.

Protocol	Architecture	Forwarding Model	Message Types	Open-Source	Version
OpenFlow [291]	Control plane physically, logically decoupled from data plane	Flow table	Feature request, packet-in, FlowMod, packet-out	Yes	v1.5
ForCES [292]	Control and data planes logically decoupled, physically together	Logical functioning block	Configuration, notification, query, response	Yes	v1.4
OpFlex [293]	Part of control plane redistributed in data plane	Policy based forwarding	Policy requests, responses, updates, withdrawals	Yes	v1.0
POF [294]	Forwarding plane oblivious of the protocol	Flow instruction sets	Configuration, packet-in, packet-out, query	Yes	v1.0
PCEP [295]	Logically decoupled control and data planes	Computed paths	Open, request, close, reply	Yes	v1.0
OpenState [296]	Decentralize some control to forwarding plane using FSM	Stateful Flow table	Packet-in, FlowMod, packet-out, StateMod	Yes	v1.0

6.2.3. Virtual Switches

A virtual switch is a software component that connects virtual machines and physical network devices, which allows network administrators to manage network topologies in a dynamic manner without having to make changes to the underlying physical network. Virtual switches have the ability to create network segments and isolate traffic between different virtual machines or groups of virtual machines, which will help improve network security. Open vSwitch and VMware NSX virtual switches are popular examples of virtual switches [420].

6.2.4. Optical Switches

Optical networks, as opposed to packet switching networks, rely on circuit switching, so KDN switches belonging to an optical network rely on optical circuit switching technology. Unlike packet switching, which operates at the network layer of the Open System Interconnected (OSI) model, circuit switching operates at the physical layer, using light paths to establish connections between the switches. This approach has lower latency, higher bandwidth, and improved efficiency compared to packet switching forwarding elements. Micro-Electro-Mechanical Systems (MEMS)-based optical switches have been used for large data center networks, which have enabled dynamic topology reconfiguration and centralized control for traffic engineering in optical networks [421]. There have been attempts to unify circuit switching in optical networks with packet switching in IP networks using a unified controller [422]. However, such unification needs to be accompanied by upgrading each physical optical switch with a virtual switch that converts control messages received from the controller to commands acceptable by the physical optical switch. Some have attempted to unify packet switching with circuit switching in optical networks by integrating a Generalized Multi Protocol Label Switching (GMPLS) control plane that

manages circuit switching with an SDN controller that manages packet switching [423]. GMPLS has been proposed as the control plane for wavelength-switched optical networks. However, a recent survey suggests that OpenFlow shows better performance in terms of less probability of blocking and average time of establishing a light path than GMPLS architecture in large multi-domain networks [424]. A photonic integrated wavelength selective switch aiming at providing flexible bandwidth for optical data center networks that can be reconfigured dynamically using the controller has been presented in [425]. In such a scheme, based on variable traffic patterns, the controller can effectively control the bandwidth per link in optical networks. Similar works suggest how photonic integrated circuits in optical switches (photonic switches) can be automatically controlled and managed using a centralized controller [426].

6.2.5. Wireless Access Points, Base-Stations, and Vehicles

A wireless access point is a networking device that allows wireless devices such as laptops and mobile phones to connect to a wired network using Wi-Fi technology, which allows multiple devices to connect and share resources [427]. Similarly, a cellular base station in a mobile communication network manages call handoffs and communicates wirelessly using radio signals with mobile devices. Vehicles in a vehicular network communicate wirelessly, either using Dedicated Short Range Communication (DSRC) for Vehicle to Vehicle (V2V) communication or using mobile communication with the aid of a cellular base station for Vehicle to Infrastructure (V2I) communication [428]. Figure 10 depicts the KDN paradigm implementation in a heterogeneous network scenario consisting of a Wi-Fi network, a cellular network, a vehicular network, an optical network, and a wired network. Therefore, for all the types of wireless communication described above, when KDN is integrated into such networks, switching can occur through an access point in a Wi-Fi network, a base station or a wireless access point in a mobile network, or a vehicle or a Road Side Unit (RSU) in a vehicular network, as shown in Figure 10.

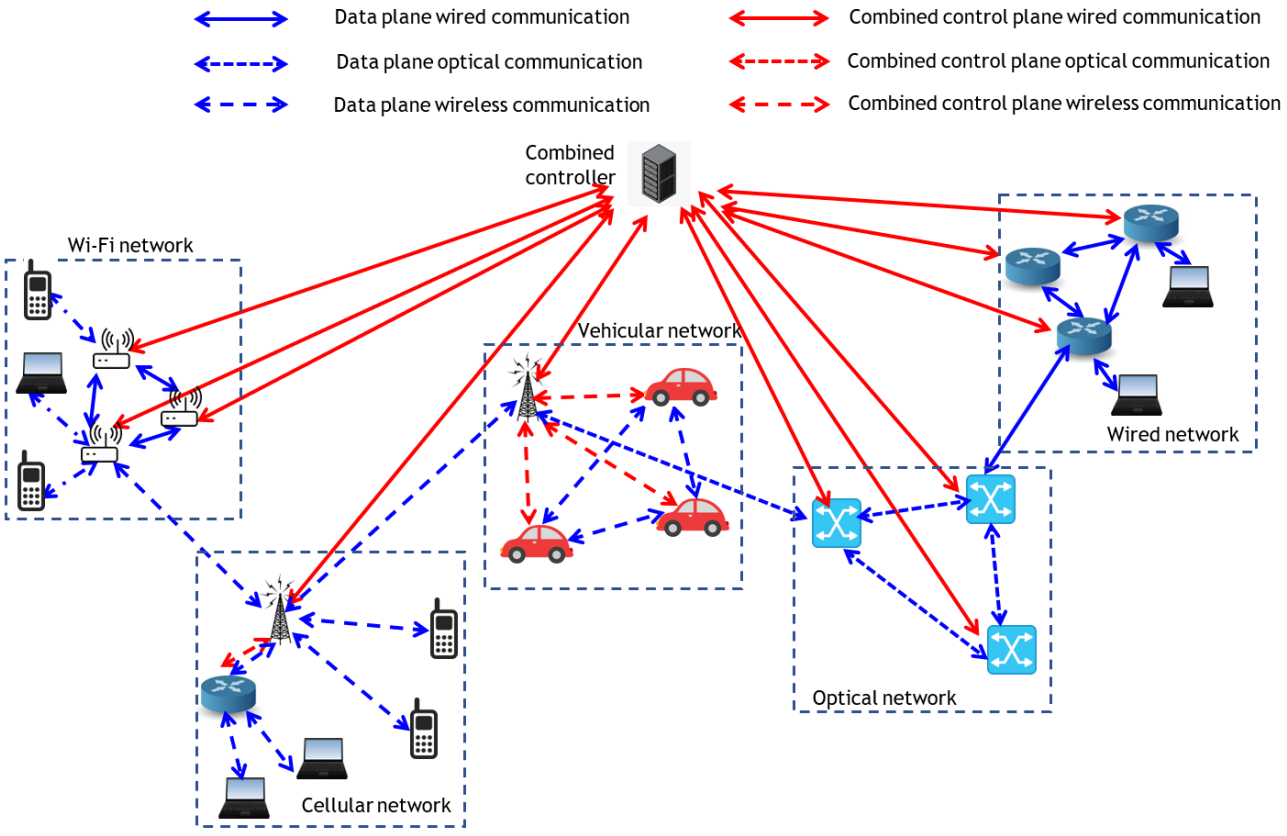

Figure 10. A KDN heterogeneous network scenario.

Note that, in Figure 10, combined controller refers to a physical entity whose management, control, and knowledge planes have been unified. However, note that, in KDN, these three planes are logically separated, even though they can be physically unified to be represented as a combined controller. Thus, wireless access points, base stations, and vehicles become forwarding devices in their corresponding wireless networks, which can be configured by a remote combined controller in the KDN paradigm. Software-Defined Radio (SDR) is the concept of controlling wireless transmission strategies in the physical layer using software [429]. Thus, SDR can be integrated with KDN to provide physical layer control of the wireless KDN switches, such as wireless access points, mobile base stations, vehicles, etc., using a combined controller. Thus, these wireless switches' physical parameters, such as link association, channel selection, transmission rate, etc., can be managed and controlled dynamically based on network statistics and network defined policies in KDN.

6.3. Transmission Media

Transmission media convey either data between the forwarding elements or data, control channel information, or management channel information between data plane elements and the control plane or the management plane. There are basically three types of transmission media: wired, optical, and wireless, which are briefly discussed in the following sections.

6.3.1. Wired Media

Wired transmission media transmit electrical signals from a given source to a given destination. The most frequently used material for electrical wires is copper. Copper electrical wires can be categorized into two types: twisted-pair cables and coaxial cables. Twisted pair cables are the most common wired medium used in LANs, consisting of insulated copper wires twisted together to reduce electromagnetic interference from other cables. In coaxial cables, a copper core is surrounded by a layer of insulation, a woven shield, and an outer jacket [430]. These cables are used in LANs as they provide better resistance to electromagnetic interference compared to unshielded twisted-pair cables. Copper wires have excellent electrical conductivity to carry signals with minimum degradation, high reliability as they can withstand harsh environmental conditions, are compatible with both analog and digital signals, and are cost effective compared to optical fibers. However, they have a lower bandwidth, are susceptible to electromagnetic interference, are susceptible to eavesdropping and hacking, etc. [431].

6.3.2. Optical Media

Optical transmission media transmit light waves with modulated data over long distances. Compared to copper wires, these provide higher transmission rates, longer transmission distances, better resistance to electromagnetic interference, and are more secure than electrical wires as they are difficult to tap and intercept [432]. The optical fiber is a thin glass or plastic strand that guides the light signal from the transmitter to the receiver. There are two main modes of optical fiber: single-mode and multi-mode. Single-mode fibers with a diameter typically less than 10 microns are used for long-distance communication to reduce dispersion and improve signal quality. Multi-mode fibers having a large core diameter are less expensive, have a lower bandwidth, have a higher dispersion, and are used for short-distance communication (up to a few kilometers) [433]. However, optical fibers are expensive to install, consume more time to repair (which increases the downtime), are more fragile than copper wires (i.e., can break easily), and are less immune to signal attenuation from light wave absorption, scattering, and reflection [434].

6.3.3. Wireless Media

Data transmission through electromagnetic waves without the need for physical wires is referred to as wireless transmission. Data can be conveyed via electromagnetic waves with a range of frequency ranges in various kinds of network configurations. For instance, Wi-Fi uses radio waves having a frequency between 2.4 GHz and 5 GHz to provide wireless internet access [435]. Frequency ranges from 700 MHz to 50 GHz are utilized in cellular networks to link mobile devices to the infrastructure [436]. Wireless media enable user equipment mobility without being constrained by wires, and, because of their great flexibility and scalability, they are more ideal for deployment in KDNs. Cellular networks are less suited to high-speed data transfer or the transfer of a huge amount of data since they have less capacity than copper lines. Furthermore, compared to single-mode optical fibers, wireless media might have a shorter communication range, inferior security due to the ease with which attackers can intercept communications, and significant electromagnetic interference from other devices [437].

Table 8 summarizes the comparison among the main transmission media used in knowledge-defined networks.

Table 8. Comparison of transmission media parameters.

Parameter	Wired	Optical	Wireless
Transmission range	Low (Cat6a—around 100 m)	High (single mode—2 km to 40 km, multi mode—300 m to 2 km)	Vary (4G—up to 10 km, 5G—up to few km, Wi-Fi6—50 m indoor and 200 m outdoor)
Attenuation	Medium (Cat6a—0.02 to 0.10 dB/m)	Low (single mode—0.0002 to 0.0004 dB/m, multi mode—0.0030 to 0.0100 dB/m)	Vary (4G—0.0005 to 0.0050 dB/m, 5G—0.0005 to 0.0200 dB/m, Wi-Fi6—0.10 to 10 dB/m)
Propagation delay	Medium (Cat6a—5.3 µs/km)	Vary (single mode—around 5 µs/km, multi mode—6.7 µs/km)	Low (4G—3.5 µs/km, 5G—3.5 µs/km, Wi-Fi6—4.4 µs/km)
Maximum throughput	Medium (Cat6a—10 Gbps)	High (single mode—100 Gbps, multi mode—10 Gbps)	Vary (4G—1 Gbps, 5G—20 Gbps, WiFi6—9.6 Gbps)
Maximum bandwidth	Low (Cat6a—500 MHz)	High (single mode—10 GHz, multi mode—2 GHz)	Vary (4G—20 MHz, 5G—400 MHz, Wi-Fi6—160 MHz)
Error rate	Medium (Cat6a—10^{-12})	Low (single mode—10^{-15}, multi mode—10^{-12})	High (4G—10^{-6}, 5G—10^{-9} to 10^{-10}, Wi-Fi6—10^{-9} to 10^{-10})
Communication cost	Low	Medium	High
Installation and maintenance cost	Low	High	High
Flexibility	Medium	Low	High
Security	Medium	High	Low
Electromagnetic interference	Medium	Low	High

7. Application Plane

7.1. Introduction to Application Plane

The application plane is the top layer in KDN architecture and is responsible for providing a higher-level view of the network that is more closely aligned with business needs and objectives. These high-level goals and objectives of users are expressed in the application plane as intents. The application plane in KDN is designed to understand the requirements of the applications, while dynamically optimizing application policies based on the knowledge of the network. Additionally, it offers a centralized control point for network policies, allowing administrators to specify policies that are exclusive to a single application or a collection of apps and to reliably apply those policies throughout the network. Remember that an application policy is a set of high-level guidelines that control how an application behaves. Its scope is more specific (application-level scope) than an intent, and it may be established for a single application or a collection of apps. Due to the fact that KDN can be implemented in a variety of network types, including

wireless sensor networks, data center networks, optical networks, vehicular networks, internet of things networks, etc., a wide range of applications may be possible. We discuss KDN in different network scenarios in Section 8.3.7. The applications can be developed by third-party vendors or customized by network operators to suit their requirements. The application plane decouples the application logic from the hardware to express the desired intents/policies in a centralized manner. Routing, load balancing, access control, firewalls, etc. are typical examples of KDN applications.

7.2. Architecture of the Application Plane

The architecture of the application plane and its interface with other planes [438] are given in Figure 11.

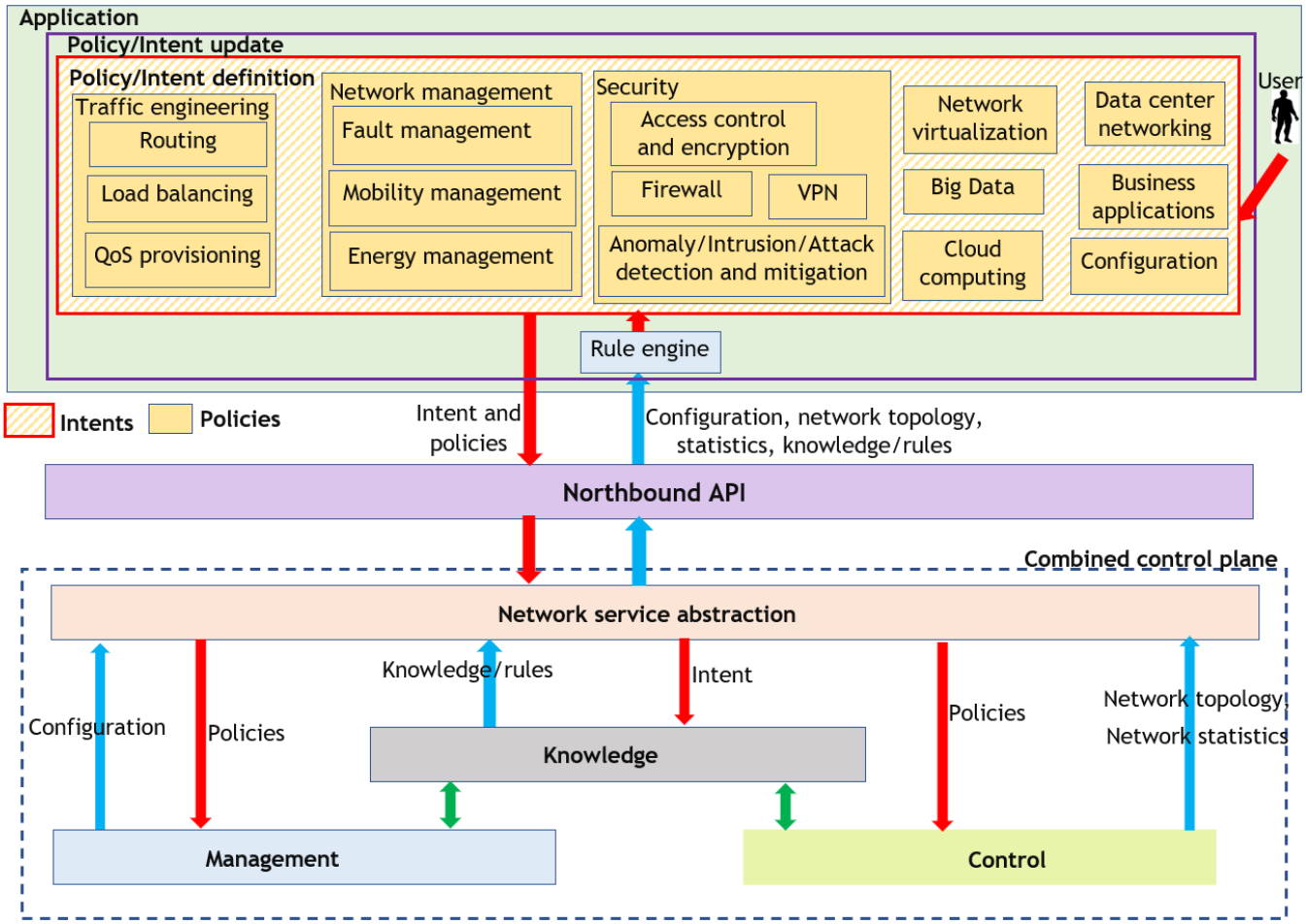

Figure 11. Architecture of the application plane, with different flows exchanged with other planes.

As evident from Figure 11, there are basically two sub-planes—the intent/policy definition sub-plane and the intent/policy update sub-plane—in the application plane. Note that the policy/intent definition sub-plane is enclosed within the policy/intent update sub-plane. The purpose of the intent/policy definition sub-plane is to define policies and intents using network administrators. Once intents and policies are defined, they can be dynamically updated using the intent/policy update sub-plane using the rules/knowledge received from the knowledge plane, the configurations received from the management plane, and the network topology and statistics received from the control plane. Thus, application policies and intents can be dynamically updated when the network state changes. Note that, in the intent/policy update plane, there is a rule engine to infer from knowledge/rules and other data received from other planes to produce decisions or instructions to update the policies. As evident from Figure 11, both of these sub-planes

enclose a set of applications. Note that some applications can be abstracted into a generic application category. Thus, policies can be specified such that the base policies of the base application are inherited by the derived (specific) applications. For instance, routing, load balancing, and QoS provisioning can be abstracted into one traffic engineering base class application. Therefore, an application policy defined for the traffic engineering base application such as "The normalized network congestion must be maintained below 50%" is inherited by all the specific traffic engineering applications such as routing, load balancing, and QoS provisioning, and it is not required to redefine the same policy in those applications. However, each specific application inherited by a base application class may have its own unique policies. For instance, the QoS provisioning application may have a policy to "Give priority to video data", which is not applicable to routing or load balancing applications.

Note that the application plane communicates with the knowledge, control, and management planes with the help of a northbound API and a network service abstraction layer. As reviewed in Section 5, the northbound API can be an interface from a network operating system, a RESTful-API, an intent-based API, or a high-level programming language. This northbound interface directly interfaces with the network service abstraction layer, which abstracts all network services and acts as the interface to communicate application requirements to other planes and convey data, knowledge/rules from management, control, and knowledge planes to the application plane. As shown in Figure 11, descriptive knowledge/rules, network topology and network statistics, and configurations flow from the knowledge, control, and management planes, respectively. In the other direction, from the application plane, intents flow to the knowledge plane, and policies flow to the management and control planes.

7.3. Application Scenarios

Applications in KDN can be broadly categorized into traffic engineering, network monitoring, security, network virtualization, cloud computing, big data, data center networking, and business applications. Each of these applications is discussed in detail in the following sub-sections.

7.3.1. Traffic Engineering

Traffic engineering is the process of optimizing the performance and utilization of network resources to ensure efficient traffic data flow. Traffic engineering applications can achieve the purpose of traffic engineering in a variety of ways, which are described in the following subsections.

Using Traffic Engineering Policies

Traffic engineering policies are a kind of set of rules that govern how traffic should be routed through the network. The traffic patterns are dynamic and can change with time; thus, traffic engineering policies should also be dynamically reconfigured considering the traffic patterns [439]. Traffic shaping policies can be used to manage the rate at which traffic flows enter or exit a network, preventing congestion and ensuring that network resources are utilized efficiently [440]. Traffic isolation policies can be used to segment different types of traffic flows from each other, improving network security and preventing malicious traffic from affecting critical applications or services [441].

Routing

Routing is different from load-balancing, which involves determining the optimal path for network traffic between two end points based on multiple factors such as the network topology and status, routing policies, routing protocol, etc. The controller is responsible for configuring the routing tables of each forwarding device by computing routes using multiple factors. Thus, routing applications provide routing policies to the control plane to use while computing routes. In [442], authors use machine learning to classify the priority of each flow to match an application-specific requirement, and then multiple paths are computed based on the application requirement. This work uses the Yen-K shortest

path algorithm in computing routes, which has increased the availability of unloaded paths for high-priority flows. Deep reinforcement learning with convolutional neural networks has been utilized to route traffic considering application-level QoS parameters in a KDN [443]. An application-aware routing scheme that considers different routing parameters for different application classes to improve the performance of all application classes has been studied in [444]. In particular, the preceding routing approach considers end-to-end delay and link load for real-time applications, delay variation and link load for streaming applications, and link load for miscellaneous applications, thus effectively computing routes suitable for each application scenario.

Load-Balancing

Load-balancing refers to the process of distributing network traffic across multiple paths or network devices to achieve better performance, higher availability, and faster response times. In a KDN scenario, basically three types of load-balancing strategies can be achieved:

- Static load-balancing: Static load-balancing refers to distributing network traffic using multiple paths using fixed rules or policies, such as routing network traffic using source or destination IP addresses, based on the type of application used, etc. However, this approach is not the ideal approach for load-balancing, as the traffic load in a real network scenario can be highly dynamic and fluctuating [445]. This approach is totally application-based, where the load-balancing is determined by high-level application policies;
- Dynamic load-balancing: Dynamic load-balancing adjusts the load-balancing rules based on real-time network conditions without any influence from any high-level policies. Dynamic load-balancing algorithms can use various metrics such as packet loss, latency, and throughput to decide on how to distribute traffic on multiple paths. For example, in [446], an algorithm for finding alternative best paths that have minimum link cost and low traffic flow is found when network congestion occurs on a certain path. Even though dynamic load-balancing may achieve better performance compared to static load-balancing, load-balancing entirely relying on real-time data may be problematic in some instances. For instance, if, for some data, the machine learning model predicts an erroneous output, then it can affect the load-balancing process negatively. However, if application policies were also involved, the effect of the erroneous knowledge could be reduced to some extent;
- Adaptive load-balancing: Adaptive load-balancing is the best approach suitable to be deployed in knowledge-defined networks, as it combines both static and dynamic load-balancing techniques to achieve optimum load distribution by considering present network status and application policies. For instance, in [447], the load status and dynamic weight of each controller are considered along with pre-defined load thresholds to adaptively achieve load-balancing. Some have used machine learning to predict link state and then used the predictions as weights to calculate the optimum path between network hosts [448]. A mechanism for load-balancing for distributed controllers that monitors the imbalance state of the entire network and uses resource consumption metrics for load-balancing to reduce communication among controllers is presented in [449]. The service-oriented load-balancing concept is a type of adaptive load-balancing scheme. In [450], data are collected in a cloud environment to measure the delay of packets, which are used to spread the workload equitably by comparing it with the round-robin scheduling policy of the applications. An application is used to classify services, and load is balanced with the objective of maximizing resource utilization and minimizing the response time of users, which is an adaptive load-balancing approach [451].

QoS Provisioning

By allocating traffic based on QoS needs, applications may ensure that their QoS requirements are met [195]. Traffic that needs high bandwidth or low latency, for instance, might be automatically routed along pathways that have been optimized for those particular needs. Applications can be guaranteed to obtain the resources they require to operate well in this way. Due to the presence of a knowledge layer, knowledge about the current network conditions can be learned in real-time to provide the required resources to high-priority traffic, such as video streams [452]. A QoS-aware flow rule aggregation that considers both the flow rule capacity of the switches and the QoS requirements of applications has been utilized in IoT networks [225]. Some have classified network traffic into application-level QoS classes using machine learning, and a path is selected for routing those classified packets based on minimum average link occupancy times or maximum average path residual capacity [453].

7.3.2. Network Management

Network management applications define policies for different management tasks such as fault management, mobility management, and energy management, which are implemented in the management plane. The management plane in KDN will consider both application policies and real-time network knowledge in arriving at management decisions for different scenarios such as fault management, mobility management, and energy management.

Fault Management

Fault management refers to the set of procedures and processes used to detect, isolate, and resolve faults that occur in the network. KDN can use knowledge generation techniques to learn from past faults and adapt the network to prevent similar issues from occurring in the future. Fault management involves several steps. First, the fault should be identified, then isolated, and, finally, the network should be restored to normal operation. The network should be continuously monitored in order to identify faults and take corrective actions in order to keep the network operational and ensure that user needs are fulfilled. In SDN, approaches such as ndb (network debugging) are used to detect causes of network failure by backtracing network events by collecting postcards received from each packet at the switches [454]. Early solutions have conventional approaches such as network checkpointing and rollback for failure recovery with the help of the centralized controller [455]. A fault detection and recovery framework called SPIDER, which detects link or node failures using stateful switches' such as OpenState periodic link probing and reroutes traffic flows in the event of failures, has been studied in [456]. Application failures can negatively affect all underlying planes and cause them to function ineffectively. A framework called LegoSDN has been proposed to recover from application failures, which has enabled fast recovery times compared to controller reboots in the case of application failures [457]. Applications can cause interference with each other, degrading the performance of the network. Thus, application interference detection and mitigation frameworks that analyze the complex interaction behaviors of multiple applications to identify and avoid unwanted network behaviors have been proposed [458]. Applications can also be used for network troubleshooting, which involves monitoring network traffic to aid network administrators in understanding the most likely network fault links so that humans can take corrective actions as a maintenance step [459]. Furthermore, there are languages such as FatTire to write network fault-tolerant programs that can specify the degree of fault tolerance required to be used alongside fail-over mechanisms [460].

Mobility Management

Mobility management applications are used to manage the movement of network users and their devices across different network domains. It should mensure that the services are continued when the users move between different domains by tracking the devices' locations to seamlessly hand off connections between different network domains such as cellular networks, Wi-Fi networks, vehicular networks, etc. Therefore, recent work

has shed light on mobility management for efficient inter-domain and intra-domain handover, which prevents packet loss and tunneling overhead to provide improved QoS to mobile users [461]. A similar network monitoring framework uses an application policy associated with fuzzy logic and a multi-path transmission control protocol for efficient handover, avoiding the ping-pong effect [462]. A Light Virtual Access Point (LVAP) is a software-based access point that runs on a virtual machine or container and provides wireless network access to client devices. LVAPs can be deployed on demand and dynamically adjusted to adapt to changing network conditions. In [463], the authors show how LVAPs can be used for efficient client handover in a global scope, which has proven to improve the flexibility of management of wireless networks. Furthermore, mobility management involves channel scheduling. An application policy that associates a channel scheduling cooperation algorithm to enable multiple access points to cooperatively control centralized downlink transmission to achieve higher system throughput and channel utilization by avoiding co-channel interference has been studied in [464]. Another important task of mobility management is the dynamic channel reservation, which dynamically allocates spectrum. In [465], the optimal number of reserved channels is decided based on secondary user retainability and channel availability, while an application policy considers the primary user channel availability minimum by monitoring the incoming traffic requests. A Hierarchical Agglomerative Clustering (HAC) framework to control all sub-channels in the network to decide on cluster merging in ultra-dense small cell networks with an application policy to mitigate severe interference among small cell base stations has been proposed in [466]. Furthermore, mobility management also includes offloading, such as Wi-Fi device to device offloading from cellular networks when device to device communication is available [467]. By monitoring fluctuating traffic loads, work in [468] uses radio resource allocation and transmit beamforming using optimization to allocate physical resource blocks, user equipment, radio units, and the downlink transmit beamforming by having an application policy to consider imperfect channel state information. Mobility management also includes radio resource sharing. A framework known as LayBack, which has an application policy to facilitate communication and computation resource sharing among different wireless technologies and operators by organizing resources into layers, uses KDN to manage fronthaul and backhaul resources, and coordinates the corporation between different wireless technologies and operators, has been studied in [469]. OpenRadio and SoftRAN are platforms that define policies to allow a software abstraction layer for decoupling wireless protocol definition from the hardware to allow sharing of MAC layers among different protocols for better handover, resource block allocation, etc. [470]. Open Radio Access Network (ORAN) is an emerging approach for designing and deploying mobile network infrastructure that aims to create more interoperable wireless networks [471]. Research in [471] suggests the integration of AI and machine learning into ORAN so that KDN and ORAN can be integrated together for efficient wireless network resource utilization and mobility management.

Energy Management

Energy management is a crucial application in a KDN that involves reducing network power consumption while attaining other infrastructure goals such as less latency, high throughput, high fault tolerance, etc., using specialized optimization algorithms and application policies. For instance, applications can use simple high-level policies such as shutting down or sleeping links or devices to reduce energy consumption [472]. Another energy-saving technique is Dynamic Voltage and Frequency Scaling (DVFS), where the manager in KDN can manage the operating voltage and frequency of the switches based on the knowledge of device workload to manage the energy of the devices [473]. Energy-aware routing has been employed with energy-aware application services such as tunneling for fast rerouting, smooth node disabling, and detection of traffic spikes and link failures, which has reduced the energy consumption of internet service providers by 5% to 35% [474]. In [475], the authors discuss the feasibility of an application that implements services in an ONOS network operating system to implement energy-aware

traffic engineering strategies. Therefore, routing and traffic engineering can also be adapted to optimize the energy efficiency of the network. Furthermore, applications can use network virtualization to save energy of the network, as it reduces the number of physical devices required. In [476], the authors use a multi-objective virtual network embedding with the objectives of minimizing network congestion and energy consumption, where they use a path service and a resource monitoring application in the manager. Network function virtualization, along with an application policy to utilize a cluster head selection algorithm in a blockchain-based distributed IoT network, has been utilized to save energy [477]. In [478], an integer linear programming optimization model is used to optimize the energy consumption of IoT nodes by activating an optimal number of network function virtualization nodes and assigning regular nodes to those activated by network function virtualization. A task assignment and scheduling platform that is formulated as a deep Q learning process strives to maximize energy efficiency by saving battery power under the constraints of application dependence, thus minimizing energy consumption considering the requirements of the applications [479].

7.3.3. Security

Security policies for access control, encryption, firewalls, attack/anomaly/intrusion detection, etc. can be defined using security applications. Applications can enforce policies to detect and mitigate threats more quickly and effectively, helping to ensure the integrity, availability, and confidentiality of network resources. KDN provides a platform to check security policies to make sure that the network receives protection by preventing security breaches.

Access Control and Encryption

Access Control (AC) mechanisms help ensure that only authorized users or devices can access network resources. Applications can define policies for access control, which will be integrated into flow rules by the controller and restrict traffic based on source and destination addresses and ports. Encryption, on the other hand, protects sensitive data from being disclosed to unauthorized parties. Some have proposed to implement access control considering network resources, security requirements, reconfiguration conflicts, etc., which implement both mandatory and discretionary AC [480]. As global-level network access control may not reflect fine-granular level access control, work in [481] suggests using user-level and flow-level access control, where access tables are initiated by network administrators and can be dynamically changed based on network activities. A controller-independent application-level dynamic access controller that mitigates application to controller threats by using 4 permission categories: read, add, update, and remove, has reduced application abuse compared to static permission control [482]. Considering the fact that malicious applications can launch hostile attacks using the northbound API, a secure application management framework to granularly manage application permissions by analyzing the legality of application permissions and using encrypted registration authorization has been studied in [483]. Mandatory access control, which employs an application policy to use extended attributes for access control based on security level, where the switch security level is regarded as the attribute of the access control environment, has been studied in [484]. This extended attributes-based AC method also has a secure path planning method based on particle swarm optimization for securing access data flow. A dynamic access control scheme to control network resources with an application policy to use broadcast encryption that deviates from updating permission lists to make sure that resources are available only to authorized users has been proposed in [485]. A decentralized access control mechanism called SILedger has been utilized based on blockchain and attribute-based encryption for effective token-based authorization of applications in heterogeneous IoT domains, where tokens are the currency of blockchain [486]. A framework known as P4-sKnock, which has an application policy aligned with a P4-based two-level host authentication and access control mechanism, where the first level introduces encrypted dynamic port knocking in order to secure the dynamic port knocking sequence by encryption, and the second level

follows an authentication measure using a challenge–response host identity verification mechanism in order to authorize, quarantine, or block the host [487]. P4-sKnock prevents man-in-the middle attacks, IP spoofing attacks, replay attacks, and provides post-port knocking authentication. An administrative model to manage Role-Based Access Control (RBAC) actions using custom fine-grained permissions to extend the capabilities of services that define the authorization of network applications has been proposed in [488]. RBAC creates custom operations to extend the capabilities of services and provide permissions to administer AC. By using smart contracts for independent, immutable, verifiable policies in blockchain, work in [489] uses blockchain for creating access control policies for IoT devices, while providing a trackable policy management system to prevent forged policy dissemination. Deviating from the static access control mechanism, Behavior-Based Access Control (BEAM) defines policies to dynamically grant permissions based on network behavior, which can upgrade or downgrade assigned access permissions at run-time [490]. BEAM further verifies and builds trust for an application. In [491], attribute-based encryption and certificate-based access control protocols are used to achieve access control, while a blockchain is used to add various transactions among controllers, applications, and switches where blocks are added using a consensus mechanism.

Virtual Private Networks (VPN)

To establish a safe connection between two networks over the internet, a virtual private network is employed. It is possible to develop an application that constructs a secure tunnel between two networks. MPLS VPNs rely on multiple protocols to function correctly, which provide security, QoS, and flexibility in networks. In [492], based on the network performance, the IPSec policy of the MPLS VPN tunnels is dynamically updated to provide better security. A secure approach called software-defined VPN, where each application is allocated its own overlay VPN and flow tables related to each VPN, are pushed by the controller to switches to improve security by separating VPN traffic and by utilizing service chaining [493]. A P4 switch-based concept for IpSec VPN, where P4 switches are served as tunnel end points for site-to-site and host-to-site applications without the exchange of complex key exchange protocols, has been presented in [494].

Firewall

A firewall is a security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules. A firewall can be implemented as an application that enforces policies to restrict or allow network traffic. In [495], the authors present a concept called pre-firewall, which is a framework to monitor firewall rules to avoid collision, redundancy, and overlapping issues, and present an algorithm to resolve rule anomalies. A virtual firewall (ACLSwitch) that filters traffic across the network at switch level using the OpenFlow protocol to distribute flow rules and defines policy domains to allow different filtering configurations to apply to different switch domains has been proposed in [496]. ACLSwitch allows subsets of switches to enforce different security policies. Similarly, in [497], a network function virtualization-based virtual firewall having policies to filter traffic with a Click module has been presented to efficiently use network hardware resources, while having equivalent performance to a firewall based on the POX controller. Another study uses network function virtualization to implement a stateful firewall, which uses a set of guidelines and rules (policies) to avoid network-hazardous connectivity [498]. A firewall called FlowTracker, which is a stateful firewall with reduced controller processing and communication overhead, uses an adaptive connection tracking policy to detect and monitor network traffic [499]. Similarly, in [500], a stateful firewall is implemented in the controller to filter traffic based on the complete context of incoming packets, having a policy to evaluate the entire context of traffic flows to filter traffic. A framework called FleXight uses per-flow dynamic sampling to convey packet-level information to the controller, which enforces a firewall application policy to detect horizontal port scans [501]. An application known as ChainGuard, which provides security to blockchains by filtering traffic to make sure that the origin of the traffic is legitimate, where illegitimate packets will be intercepted by the firewall [502]. ChainGuard is capable

of mitigating flooding attacks and providing access control functionality. Considering the fact that firewall policies need to be changed when the types and sources of attacks change, a computationally efficient firewall Rule Update Algorithm (RUA) to compute the rules related to the updated policies while preserving non-updated policies is presented in [503]. RUA is formulated as an Integer Linear Programming (ILP) problem to satisfy rule protection and TCAM constraints with the objective of minimizing wasted bandwidth and the number of rule placements. A firewall called P4Guard, which is a protocol-independent, platform-agnostic software-based firewall that is configurable using a high level domain-specific language to specify policies in order to specify packet processing logic using P4 [504].

Anomaly/Attack/Intrusion Detection and Mitigation

Intrusion/anomaly/attack detection and prevention systems monitor network traffic for signs of malicious activity and take actions to prevent it. These systems look for signs of suspicious behavior or attempted attacks, which is different from the approach taken by firewalls, which filter traffic based on a predetermined set of rules. These systems will use one of signature-based detection, behavior-based detection, or anomaly-based detection to detect attacks. Signature-based detection involves comparing network traffic to a database of known attack signatures, while behavior-based detection looks for patterns of activity that deviate from normal behavior. Anomaly detection uses statistical analysis to identify traffic patterns that are unusual or unexpected [505]. Once attacks are detected, KDN can install packet forwarding rules on switching devices to block the threat. The application policy will determine which type of detection out of signature-, anomaly-, or behavior-based detection will be utilized in the IDS.

A behavior-based Intrusion Detection and Prevention System (IDPS), with an application policy to specifically detect port scanning using the port bingo algorithm and denial of service attacks using flow statistics, has been investigated in [506]. This behavior-based IDPS uses two rate-based connection monitoring algorithms, which are Credit-Based Threshold Random Walk (CB-TRW) and Rate-Limiting (RL). In [507], the authors analyze the performance of different machine learning techniques for an application that uses anomaly-based intrusion detection to identify the classifier, which leads to a more secure network. An intrusion detection system with an application policy to use signature-based intrusion detection using a random forest classifier to detect and prevent denial of service attacks has been studied in [508]. An anomaly-based intrusion detection system has been used, which has an application policy to identify the attack type and the source of the attack in a resource-constrained wireless network environment using an online change point detector to monitor performance metrics that are impacted when the network is under attack [509]. In collaborative intrusion detection, several intrusion detection nodes detect intrusion at each node while mutually exchanging information such as attack signatures, attack alarms, etc. between each other. A relatively new approach called Challenge-Based Collaborative Intrusion Detection Networks (CBCIDN) is a detection framework with a policy for identifying malicious nodes by calculating the nodes' reputation by sending a special message called a challenge message to the nodes [510]. CBCIDN is capable of detecting newcomer attacks and betrayal attacks to quickly identify the malicious nodes. In a distributed KDN control architecture, an intrusion detection system having an application layer policy such that the controllers jointly train a deep learning and generative adversarial networks-based global intrusion detection model without exchanging sub-network flows for a vehicular ad hoc network has been investigated in [511]. Such a collaborative IDS has been shown to be effective in both Independent Identically Distributed (IID) and non-IID scenarios. Furthermore, for collaborative intrusion detection to have secure communication between each intrusion detection node, application policies have been defined to use blockchain for establishing trust-based communication among detection nodes to propose a Snort-Based Collaborative Intrusion Detection System (SBCIDS) [512]. SBCIDS receives the latest updates from the controller to securely share the signatures among the Snort nodes using the blockchain. A self-learning network intrusion detection system having an

application policy to use Hidden Markov Models (HMMs) in a KDN, which can monitor the whole network and learn from evolving network activity adaptively to react to intrusion detection events, has been presented in [513]. Similarly, there have been numerous recent research attempts at the implementation of self-learning intrusion detection systems in KDN scenarios that have policies to use the prospect of machine learning to adapt and evolve over time [514].

7.3.4. Network Virtualization

Network virtualization is the process of creating multiple logical networks on top of a physical network infrastructure using network slicing, which allows multiple networks to coexist and share the same physical resources, while each logical network appears as a separate entity with its own network policies and configurations. Network virtualization allows either network-level slicing that slices the physical network into multiple virtual networks or flow-level slicing that slices the physical network based on different flows. FlowVisor is the technology used for achieving network slicing by splitting the forwarding plane of the networking devices into multiple slices, each with its own independent set of rules and forwarding policies, which allows multiple tenants to securely share the same infrastructure while maintaining their own isolated network topologies, routing, and forwarding policies. Network virtualization allows multi-tenancy, which allows service providers to deploy services over a single physical infrastructure in an isolated and cost-effective manner, separated from services provided by other service providers [515]. Multi-tenancy makes the network flexible in time, space, and the services provided. In [516], FlowVisor is used for network slicing to improve network security by creating isolated virtual networks. FlowVisor provides a centralized point of control for the network, allowing the administrator to monitor and manage the traffic and resources of all the virtual networks. This makes it easier to experiment with new network protocols, test new applications, and isolate network traffic for security purposes without affecting the rest of the network. Similarly, in [517], VLAN tags are used instead of MAC addresses to identify users with an application policy to slice a network based on roles using an authentication controller, which has resulted in a lower flow setup latency. VeRTIGO is an extension of FlowVisor that allows the controller to set the level of virtual network abstraction, allowing dynamic and advanced network slicing [518]. VeRTIGO contains policies for more granular control over network resources, enabling finer-grained management of network traffic and better isolation between tenants. Additionally, VeRTIGO's architecture is designed to be more scalable and flexible, allowing for easier management of large-scale networks with high levels of traffic and complexity. A Virtual Tenant Network (VTN) can be used to define policies for flow-based slicing, which enables network operators to prioritize and segment traffic according to specific criteria such as application type, user identity, or Quality of Service (QoS) requirements, ensuring optimized network performance and security. An application having a routing planning mechanism and a bandwidth resource planning mechanism has been utilized to satisfy VTN user requests, which has increased the efficiency of routing assignment and bandwidth utilization [519]. A multipath network virtualization scheme using policies for network function virtualization that obtains a summary of network resources to select and spread flow over multipaths, where network virtualization provides computing and storage resources for flow splitting, packet reordering, etc., has been studied in [520]. A network virtualization framework called Open Network Hypervisor (ONVisor) has been designed with policies to provide isolated control and data planes per virtual network, support for distributed operators, on-platform virtual network application development and execution, and support for heterogeneous data planes [521]. By considering the shortcoming of efficiency in distributing physical network resources among virtual tenants in the OpenVirtex network virtualization platform, an application for a QoS management mechanism using a resource manager to distribute token buckets among virtual networks according to their weights is proposed in [522]. Recently, researchers have suggested adding applications with policies to integrate blockchain technology for network virtualization to

protect owners of wireless infrastructure from double-spending attacks that allocate the same radio frequency slice to multiple virtual wireless networks [523].

7.3.5. Big Data

Big data refers to extremely large (in volume, velocity, and variety) and complex data sets that cannot be easily managed or processed using traditional data processing methods. The ability to analyze and make sense of big data has become increasingly important in various fields such as healthcare, business, scientific research, etc. KDN has been helpful in solving issues that exist with big data applications such as data processing in cloud data centers, data delivery, scheduling, etc. by efficiently managing the network [524]. On the other hand, big data can help KDN with traffic engineering, mitigation of security attacks, cross-layer design, etc. An efficient workload slicing scheme processes big data in order to handle data-intensive applications in a multi-edge cloud environment, where inter-data center data transfer is handled by an application policy based on an energy-aware traffic flow scheduling technique [525]. A real-time big data streaming framework called Typhoon, which operates based on a policy of partially offloading application layer data routing and control to the network layer in KDN in order to achieve on-the-fly programmability and high-performance data routing, has been studied in [526]. A big data simulation tool to integrate big data applications with SDN is called BigDataSDNSim, which contains the big data management system: Yet Another Resource Negotiator (YARN) and SDN-enabled networks in a cloud computing environment has shown superior performance in Map-Reduce applications [527]. BigDataSDNSim implements application policies such as the Map-Reduce application selection policy and the Hadoop Distributed File System (HDFS) replica placement policy. A service called AmoebaNet has been proposed to overcome limitations existing in big data science, such as the last mile problem, scalability, and programmability problems, by applying KDN to provide QoS-guaranteed services [528]. AmoebaNet allows application programs to program networks at runtime to optimize for optimum performance. With the help of big data analytics applications, by using multi-dimensional analysis of key performance indicators and machine learning to generate knowledge regarding traffic congestion, QoS can be managed in knowledge-defined networks [529]. A web service manager called AWESoME is an application that identifies and prioritizes the traffic of important web services by using big data algorithms to build models describing the traffic of a large number of web services while installing flow rules related to the services [530]. In KDN, big data can be analyzed using machine learning techniques for clustering, forecasting, and managing traffic behaviors with the aid of big data applications that define policies for network function virtualization [531]. Big data, machine learning, KDN, and application policy-driven network function virtualization can be integrated together to build a framework for self-organizing networks and for network slicing [532].

7.3.6. Cloud Computing

Cloud computing refers to the delivery of on-demand computing resources, including servers, storage, applications, and services, over the internet. In a cloud computing environment, by designing cloud computing applications with policies for network function virtualization, network functions can be achieved by utilizing cloud computing resources [533]. A framework that integrates cloud computing platforms with network orchestration and compares single controller orchestration with orchestration with application based network operations has been studied in [534]. An optimal resource allocation and virtual machine placement model for multi-tier applications in large cloud data centers that optimizes the data center's energy and communication costs that influence the cloud's performance has been investigated in [535]. In [536], QoS levels for each application are defined, and queuing for different application flows is scheduled at the switches for end-to-end QoS-guaranteed routing for cloud applications and services. A cloud-enabled secure IoT architecture called CENSOR, which uses a scalable software remote attestation

scheme that ensures integrity to achieve application-specific goals in the network, has shown improved data communication and reduced overhead [537]. Cloud storage can be secured by using cloud computing applications that have policies integrating blockchain to create a deep trust between the transaction levels and the controller in KDN, which provides reliability and versatility [538]. Cloud computing can be useful in the event of natural or man-made disasters for providing rapid situational awareness. Driven by such an application, work in [539] proposes an incident-supporting visual cloud computing solution to collect, compute, and consume at the network edge, coupled with cloud offloading to a core computation using knowledge-defined networking.

7.3.7. Data Center Networking

Data center networking is the practice of connecting servers, storage devices, and other computer resources within a data center. Data centers should be designed to provide resources based on application requirements to provide services with less latency, intelligent resource utilization, and improved efficiency. A Highly Efficient Switch Migration (HESM) for load-balancing in an optical data center network, where switches are selected by minimizing migration costs, has been studied in [540]. Applications define policies for HESM, use multiple load metrics to measure the load of controllers, and select the optimal target controller with the largest remaining resource. An application with a controller-proxy method where the controller can dynamically delegate portions of event processing back to data plane switches to reduce the workload of the controller has been proposed for large-scale data center networks [541]. A congestion control technique for large data center networks that has an application policy to allocate more bandwidth to burst flows and reduce the bandwidth of background flows when switch congestion is detected has been studied in [542]. An effective energy management application by switch-on–off policy and an effective rerouting policy in a data center network that has achieved higher energy efficiency is presented in [543].

7.3.8. Business Applications

Business applications are typically revenue-generating applications that define policies that help organizations achieve their business goals. In [544], an intent-based northbound interface with micro-services, and service-oriented design principles having a three-tier application architecture and domain-driven design is proposed to define business-like applications. A framework known as FlowVista, which identifies flows using the Northbound interface, where flows are matched directly by interacting with a business application, translates high-level business policies into low-level network flows [545]. Voice Over IP (VoIP) is a very good example for the realization of FlowVista, as VoIP is driven by high-level business policies. There has been a need for KDN-based high-performance computing solutions for enhanced throughput and better utilization of bandwidth to meet business needs [546]. Enterprise Integration Pattern (EIP) is a combination of design patterns that combines existing and new business applications in an enterprise environment. A communication framework that enables communication between applications in an enterprise environment and creates virtual local area networks using EIP by integrating with KDN to improve programmability and global view has been studied in [547]. Similarly, in an enterprise network, a context-aware communication framework using EIP to offer host anonymity by replacing IP addresses with spoofed IP addresses and achieve context awareness through KDN global visibility, where business application services route traffic based on business service requirements rather than network layer information [548].

7.3.9. Configuration

The application layer has network administrator-defined policies for network configuration. Note that, in KDN architecture, these policies can be dynamically updated based on the current network status or performance received by the application plane in the form of descriptive knowledge or rules and based on current network configuration information received from the management plane. This configuration application defines policies for network backup and restore, compliance, logging and auditing, error handling, firmware configuration, authentication, authorization, encryption, etc. The backup and restore policy defines the frequency of backups, the location of backups, and the procedures for restoring a switch configuration [549]. The compliance policy specifies the set of rules and regulations that protocols such as OF-CONFIG should comply with, such as the OpenFlow specification, security best practices, and industry specific-regulations [550]. The logging and auditing policy specifies the types and events to be logged, the format of logs, the retention period for log files, etc. The error handling policy specifies the procedures for handling errors, such as error codes and messages that are used to indicate failures and errors during configuration and management, retrying operations, reporting errors to the administrator, etc. The authentication policy specifies the allowed authentication mechanisms and the required credentials for providing access. The authorization policy specifies the types of operations that are allowed or denied based on user roles, privileges, and permissions [551]. The encryption policy specifies the encryption algorithm and the key length to be used for securing communication. The firmware configuration policy specifies the firmware version that each device should run; the firmware upgrade policy specifies the schedule for upgrades, target devices for upgrades, and the criteria for determining whether an upgrade is necessary, as well as the policies for rollbacking firmware upgrades in case of compatibility issues [552]. These policies will help configure the network swiftly to provide better network management with minimal configuration problems.

Table 9 summarizes the details of different applications of the application layer.

Table 9. Network application scenarios.

Group	Sub-Group	Name	Purpose	Controller	Performance
Traffic Engineering	Policies	Roadmap [439]	Reconfiguring policies for traffic patterns	SOX, Maestro	Discuss performance of traffic engineering
		Sincon [440]	Traffic shaping	ONOS, POX	CC throughput, forwarding times increased by 3.8, 2.86
		TSCH [441]	Traffic isolation	—	High reliability in best effort traffic
	Routing	AMPS [442]	Application specific multipath flow routing	AMPS	High availability of high priority flow unloaded path
		DRL-QoS [443]	Application QoS aware routing	——	Improved routing performance in complex networks
		App-RS [444]	Application based routing parameters	Floodlight	Better performance for all application classes than CORouting
	Load-balancing	DLB [446]	Dynamic load-balancing considering congestion	Floodlight	High throughput, low delay, packet loss in fat-tree DCN
		ELB-MC [447]	Controller load status, weight based balancing	Floodlight	Low communication overhead in control plane
		ALBLP [448]	Use link state prediction to compute paths	Ryu	23.7%, 11.7% better than OSPF, Q-learning
		Aloba [449]	Use resource consumption metrics	Floodlight	Low overhead while reaching network requirements
		S-ICM [450]	Uses measured network delay of packets	Onix	Better at avoiding system saturation than HFA, RR
		SBLB [451]	Maximize resource utilization, minimize user response time	Floodlight	Significant reduction in average response, reply time

Table 9. *Cont.*

Group	Sub-Group	Name	Purpose	Controller	Performance
Traffic Engineering	QoS provisioning	RL-QoS [195]	Select routing algorithm for QoS traffic flow	Floodlight	Best trade-off between QoS vs. QoE
		ARVS [452]	2 QoS levels for base, enhancement layer packets	Floodlight	Reduce 77.3% PLR, 51.4% coverage against OpenQoS
		Q-flag [225]	Flow rule aggregation based on application QoS	POX	Reduce average delay, QoS-violated flows by 22%, 30%
		QAR [453]	Finds paths based on application QoS requirements	—	Better blocking probability, dimension reduction
Network management	Fault management	NetRevert [455]	Network checkpointing and rollback	—	Improvements on affected delay during failure
		SPIDER [456]	Detects link failures using stateful switches	Ryu	Losses are always lesser than OpenFlow
		LegoSDN [457]	To recover from application failures	Floodlight	Recover failed apps 3× faster than controller reboot
		MSAID [458]	Application interference detection, mitigation	Floodlight,ONOS	Detect known and unknown interference within minutes
		RADAR [459]	Traffic monitoring for network troubleshooting	OpenDaylight	Reduced cost and ties into business metrics
		FatTire [460]	Write fault tolerant programs	—	Able to respond extremely rapidly to failures
	Mobility management	MM [461]	Inter- and intra-domain handover	—	Improved handover and resource utilization efficiency
		SVHAF [462]	Fuzzy logic, MPTCP for handover	—	No ping-pong effect, high QoS, low undesired handover
		APV [463]	Using LVAPs for handover	—	Handover resulted less significant delays or packet losses
		CCT [464]	Channel scheduling cooperation for downlink	Ryu	Good throughput, channel utilization, packet loss rate
		DCR-CRN [465]	Decide optimal number of reserved channels	—	Reduced SU cost, network unserviceable probability
		HAC [466]	HAC to mitigate interference	—	Improves system payoff, less interference, high SE
		WiFi-D2D [467]	Wi-Fi device-to-device offloading	—	Improved offloading performance
		JRRA-BO [468]	Radio resource allocation, beamforming optimization	—	High achievable sumrate, low power consumption
		LayBack [469]	Resource sharing among wireless technologies	—	Increased revenue rate compared to CRAN
		Openradio [470]	Decouple wireless protocol definition from hardware	—	Can modify the PHY, MAC layers to implement protocols
	Energy management	ECCKN [472]	Policies to shutdown/sleep links, devices	—	Improved network lifetime, number of alive, solo nodes
		DVFS [473]	Dynamic Voltage and Frequency Scaling	—	Low power consumption at any operating frequency
		SENAtoR [474]	Energy aware routing and services	Floodlight	Reduce energy consumption by 5% to 35%
		GETB [475]	Services to test energy aware traffic engineering	ONOS	Provides a platform to evaluate energy efficiencies
		CHS-NFV [477]	Save energy using blockchain, NFV, CHS algorithm	Floodlight	High throughput, low response time, gas consumption
		ILP-NFV [478]	Optimize energy consumption using NFV nodes	Contiki	Reduced communication energy consumption, high PDR
		EATS-DRL [479]	Application based task scheduling to reduce energy	Ryu	Upto 50% less time delay, 87% energy saving

Table 9. *Cont.*

Group	Sub-Group	Name	Purpose	Controller	Performance
Security	Access control	AC [480]	Resources, security, reconfiguration conflicts aware AC	——	Easier to detect and resolve conflicts
		AC-Integrate [481]	User and flow level AC	——	Design and feasibility of a global AC system is studied
		Controller-DAC [482]	Application level dynamic AC	ODL, ONOS, FL, Ryu	Prevent API abuse—less than 0.5% performance overhead
		SEAPP [483]	Secure application management framework for AC	Floodlight	Effective security with negligible overhead
		E-ABAC [484]	Extended attributes based AC	POX	Effective access control with little impact on response time
		BENBI [485]	Dynamic AC using broadcast encryption	Floodlight	Scalable as cost is independent of SDN entities
		SILedger [486]	Decentralized AC using blockchain, encryption	——	Effective AC with negligible overhead
		P4-sKnock [487]	P4-based 2-level host authentication, AC	ONOS	Authenticate a new host within 500 ms, prevent attacks
		RBAC [488]	Role-based AC using custom permissions	Floodlight	Feasibility of custom permissions enabling AC is studied
		SC [489]	Use blockchain for creating AC policies	SDN-WISE	Node size does not affect resource access delay, throughput
		BEAM [490]	Behavior-based AC	——	No performance analysis presented
		PBAC [491]	Encryption, certificate based AC using blockchain	——	Prevent several potential attacks, better performance
	VPN	MPLS-VPN [492]	Update IPSec policy of the MPLS VPN tunnels	——	Results in QoS improvement
		SD-VPN [493]	Each application is allocated with own overlay VPN	OpenDaylight	Improves scalability of overlay VPN, security, low cost
		P4-ipsec [494]	P4 switch-based concept for IpSec VPN	ODL, ONOS	Security use cases have benefit from P4
	Firewall	Pre-Firewall [495]	Monitor firewall rules to avoid collision, redundancy	Floodlight	No performance evaluation presented
		ACLSwitch [496]	Filter traffic at switch domains	Ryu	Control network behavior holistically, compartmentally
		FlowTracker [499]	A stateful firewall with adaptive connection tracking	POX	Low controller processing and overhead, latency increment
		SFE-SBS [500]	Context aware stateful firewall	POX	Distributed controllers perform better than centralized
		FleXight [501]	Detect horizontal port scans	——	Detects 99% susceptible victims with 0.75% overhead
		ChainGuard [502]	Filter traffic based on source legitimacy for BCs	Floodlight	Mitigate DoS and DDoS flooding attacks
		DRP [503]	Firewall rule updating policy using ILP	Ryu	Less computation time to have optimized rule placement
		P4guard [504]	Protocol independent, platform agnostic P4 firewall	ClickOS	Faster packet processing time, lower network latency

Table 9. *Cont.*

Group	Sub-Group	Name	Purpose	Controller	Performance
Security	Anomaly/ Attack/ Intrusion detection	IDPS-PB [506]	Behavior-based intrusion detection:port scanning, DoS	POX	False positive rate decrease with threshold increment
		ABID-ML [507]	Anomaly-based intrusion detection using ML	—	Decision tree ML classifier has the best performance
		EID [508]	Signature-based intrusion detection for DoS	—	Classification accuracy reaches 99.7% for random forest
		CDID [509]	Attack and source identifying anomaly detection	—	Identifying probability 0.89, 0.93 for centralized, distributed architectures
		CIDN [510]	Challenge-based collaborative intrusion detection	POX	Resist the newcomer attack and betrayal attack
		CIDS [511]	Collaborative intrusion detection using ML	—	Good precision, recall, accuracy, F1-score
		SBCIDN [512]	Blockchain for trust-based communication in CID	Ryu	96% true positive rate for detection
		HMM-NIDS [513]	Self-learning IDS using HMM	Floodlight	Packet flagged percentage for different datasets are shown
		ML-NIDS [514]	Self-learning IDS using GRU-RNN	POX	89% and 99% accuracy for NSL–KDD, CICIDS2017
Network virtualization	—	FS-VANET [515]	Multi-tenancy in multiple service provider networks	—	No performance analysis is presented
		FlowVisor [516]	Network slicing using MAC	POX	Results on tenant isolation is presented
		FlowVisor-RB [517]	Role-based network slicing using VLAN tags	Floodlight	Flow setup latency reduced by 14% to 60%
		VeRTIGO [518]	Advanced and dynamic network slicing	—	No performance analysis presented
		DRA-VTN [519]	Routing and resource planning in a VTN	—	High routing assignment efficiency, bandwidth utilization increment
		MNFV [520]	Multipath network function virtualization	—	Superior performance wrt. load balancing
		ONVisor [521]	Network virtualization framework	ONOS	Low control plane, similar data plane performance
		T-AQoS-VN [522]	QoS control in virtual networks	—	1.6 times performance improvement vs. other approaches
		ECB-WNV [523]	Secure network virtualization with blockchain	—	High per user throughput when network is overloaded
Big data	—	SDB–BDS [524]	Management of big data	ONIX,POX,Ryu,FL	Use cases are discussed
		ODM-BD [525]	Decision making for big data processing	—	Low energy consumption, delay, cost
		Typhoon [526]	Big data streaming framework	Floodlight	Better throughput, management of running applications
		BigDataSDN-Sim [527]	A simulator for analyzing big data applications	Floodlight	Improve performance of MapReduce applications
		AmoebaNet [528]	QoS guaranteed network service for big data	ONOS	Solves last mile and scalability problems in big data
		BD-QoS [529]	Big data technologies to manage QoS	POX	Used ML to discover correlations, make predictions
		AWESoME [530]	A web service manager application for BD	—	Scalable, negligible load, reproducible
		BD-ML-NFV [531]	Apply ML, NFV for big data traffic clustering	—	High accuracy in traffic clustering, forecasting
		BD-ML-NFVS [532]	Apply ML, NFV for big data network slicing	—	High accuracy in traffic classification, took 5.7 min

Table 9. *Cont.*

Group	Sub-Group	Name	Purpose	Controller	Performance
Cloud computing	—	NFV-C [533]	NFV in cloud computing	—	Reduce server response time, improve QoE
		CC-O [534]	Orchestration integration with cloud computing	OpenDaylight	Setup delay time varies in range 2.5–5 s
		ENA-VM [535]	Energy, apps, network aware VM placement	—	Minimizes energy consumption, placement cost, communication traffic
		CENSOR [537]	QoS guaranteed routing for cloud applications	—	Feasible and effective for a SmartCity usecase
		Blk-sdotcloud [538]	Blockchain-based security for clouds	OpenDaylight	Better throughput, response time, fast file transformation
		IS-VCC [539]	Incident supporting visual cloud computing solution	—	High QoE, throughput, low latency, congestion
Data center networking	—	HESM [540]	Switch migration for controller load balancing	Ryu	Reduce migration cost, improved load-balancing
		Cont-proxy [541]	Network management of data centers	Floodlight	Reduce controller workload, improve scalability
		SDTCP [542]	Congestion control of large data centers	Floodlight	Burst flows: High tolerance, Short flows: better flow completion time
		ERM-DC [543]	Resource management application in data center	ODL, POX, NOX	77%, 37%, 63%—energy saving, RT delay, bandwidth utilization
Business	—	IBA [544]	Intent-based business applications	ONOS	Appropriate, effective, realizable, practical for business applications
		FlowVista [545]	Translate business policies to network flows	OpenDaylight	Portability, low bandwidth, integratability
		HPC-BE [546]	High performance computing for business environment	—	60–80% bandwidth utilization, 45–60% throughput
		EIP-CF [547]	Enterprise integration pattern communication framework	Ryu	Improved security, efficiency, reliability of enterprise networks
		CACS [548]	Context aware communication framework for business	Ryu	Protects anonymity, computationally complexity is O(N)

8. Discussion

As reviewed, many researchers have recently generated knowledge to aid in making network decisions. Even though the architecture of KDN was proposed by researchers in [27] about two decades ago, it had not gained recognition due to the technology gap until very recently, when machine learning techniques had been well developed and hardware resources could cope with the demanding resources of KDN. According to our best knowledge, we are the first to conduct a survey on knowledge-defined networking. In this section, we discuss the benefits, challenges, design guidelines, and ongoing research of KDN.

8.1. Benefits

Compared to SDN, KDN architecture brings many benefits, such as enhanced automation and intelligence, higher adaptability, reduced downtime, improved security, etc., which are discussed below.

8.1.1. Enhanced Automation and Intelligence

KDN typically uses ML to learn from data, making it more intelligent and efficient. This intelligence allows for higher automation, reducing human intervention in the management of networks as much as possible. Management of conventional networks has been achieved using a manual approach, where network administrators define static configurations for the network devices using a CLI. Such manual approaches are error-prone,

consume a lot of time, and cannot dynamically adapt to network changes. SDN provides a centralized platform for network management and makes it more easier than a manual approach; however, SDN network management is based on static application policies. KDN can automate network management by dynamically updating application configuration policies based on knowledge of network status to automatically configure the network without human intervention. KDN learns from large amounts of data to identify patterns and anomalies, allowing the network to automatically adapt and optimize network operations. For instance, the KDN can automatically adjust network resource allocation based on traffic patterns to ensure that critical applications receive more resources. Maintenance can also be automated using the concept of predictive maintenance, which uses the data collected about device statistics, event logs, etc. as input to the machine learning models to predict whether maintenance is required or not. If maintenance is required, network administrators can take proactive steps for maintenance before the issues become problematic. Repetitive network management tasks such as device configuration and software updating can be automated using KDN, where the network will be monitored using the data collected from the devices and configured when required without human intervention.

8.1.2. Higher Adaptability

KDN has a more flexible network architecture that can easily adapt to changing network environments, which includes the ability to quickly add or remove network devices and services and the ability to scale network resources up or down as required. KDN can analyze vast amounts of data, identify patterns, and adapt network operations accordingly. One way of achieving adaptability is by automatically adjusting network resources based on knowledge generated by analyzing traffic patterns using machine learning. In such a case, for instance, if the knowledge plane detects very high traffic, the management plane can allocate more resources such as energy, bandwidth, etc. to make sure that the network operates swiftly without performance issues. After some time, when it detects low traffic, resource allocation can be dynamically reduced. Another example that can be obtained from fault management is the ability of KDN to adapt to link failures, where in such cases, the traffic can be rerouted in alternative paths, ensuring that the network operates with minimal interruptions. On the other hand, when new devices are introduced into the network, these devices can be automatically configured using the management plane, and network policies in the application plane can be adjusted to accommodate these devices. Furthermore, KDN can learn from data to generate knowledge or rules to adjust network policies, optimizing network performance over time. In this manner, it can dynamically adapt to network changes in real time.

8.1.3. Reduced Downtime

In fault management, machine learning can be used to analyze traffic patterns to predict potential device or link failures even before they really occur, so that network administrators can prevent such failures by proactively attending to affected devices either by repairing or replacing them. Moreover, KDN can automatically detect network faults such as malfunctioning of devices by analyzing patterns of data to generate knowledge on device functionality so that malfunctioning devices can be isolated. Another example is detecting malicious activities of network devices, where KDN can in realtime analyze traffic patterns, security logs, etc. to detect any anomaly in a network device and can take necessary actions. For instance, if malware is detected on a device, KDN can take action to immediately isolate such devices to prevent it from spreading to other parts of the network. In case of such device/link failures or device anomaly detection, KDN can automatically reroute traffic through alternative paths or automatically switch to backup devices or links, making sure that the network operates with fewer interruptions and minimizing the impact of failures or intrusions on network operations.

8.1.4. Improved Security

Using network data, KDN can learn, using either machine learning or any other heuristic model-based method, the behavior of normal network traffic and traffic with anomalies. When an anomaly is detected, it indicates a security threat. For instance, if there is an unusual amount of traffic coming from a user, such suspicious users can be isolated or reported to network administrators to make decisions. Additionally, if KDN uses machine learning to identify a malware attack, it may instantly isolate the affected device to stop it from spreading. To train machine learning models to accurately identify unknown assaults, they can be synthesized from data from existing attacks using semi-supervised learning techniques such as variational autoencoders, generative adversarial networks, etc. This allows the detection of unknown attacks by KDN systems that otherwise may have gone undetected in SDN systems. Furthermore, using the prospect of knowledge plane, KDN can be used to detect different types of attacks, such as DDoS attacks, TCP–SYN attacks, ICMP flood attacks, malware attacks, spoofing attacks, injection attacks, etc. allowing the detection of a vast variety of attacks, where appropriate actions can be taken based on the type of attack detected, thus improving the overall security of the network. Therefore, KDN's machine learning algorithms allow detection and response to such threats more effectively than traditional network security measures.

8.1.5. Simplified Network Management

KDN can use ML to automate many of the routine tasks involved in managing the network. KDN architecture has a logically separated management plane dedicated to network management tasks. For instance, the KDN management plane can dynamically allocate bandwidth, allocate spectrum, schedule channels, offload data, etc. by automatically analyzing the mobility patterns of wireless devices and considering appropriate mobility management policies. KDN can automate the process of configuring network devices, which can save time and effort of network administrators. When there are new devices in the network, KDN can automatically detect and configure them without human involvement. It can further simplify troubleshooting by providing network administrators with real-time insights into network performance and potential issues, which will help them identify and resolve network problems. KDN can use predictive analytics, that is, predicting problems in the network even before they occur, using data analysis techniques to take proactive measures to prevent them. KDN allows centralized network management, which enables network administrators to manage all network devices from a single location, similar to SDN. However, the management in KDN is more simplified than in SDN, as routine tasks such as device configuration can be automated, while the configuration policies and management policies will have to be defined by the network administrators only once, and the defined policies can be dynamically updated using the knowledge or rules generated from the knowledge plane based on the network status to provide adaptive network management.

8.1.6. Better Network Performance

As discussed in the policy enforcement section, a policy engine can be used to convert high-level policies into rules by considering the policy requirements and in-network knowledge. Consider a network monitoring task that involves monitoring the resource utilization of network devices. KDN can improve resource utilization by identifying and eliminating network inefficiencies, which include identifying underutilized network resources and reallocating them to areas of the network that require resources. Network performance can be improved by real-time network monitoring for fault management, where faults in the network can be detected in real time using fault management policies and in-network knowledge, which can help network administrators quickly identify and troubleshoot network issues. Predictive network analysis can be used to predict events even before they occur, such as network congestion, possible device failure, loss of communication with the base station, etc., where early actions can be taken to reduce undesired effects. For

instance, by predicting whether a mobile device is likely to have a loss of communication with the base station, a handover mechanism can be activated to assign the mobile device to an available base station before the loss of communication with the current base-station occurs, such that the mobile device's communication loss time is minimized. KDN can automatically optimize network infrastructure based on network usage patterns, which includes identifying bottlenecks and adjusting the network resources to ensure that traffic flows smoothly. KDN can shape traffic in real time by dynamically allocating resources based on traffic patterns, which will ensure that critical applications have the necessary resources such as energy, bandwidth, etc. to operate efficiently and a high priority in forwarding traffic, while non-critical traffic is deprioritized.

8.1.7. Enhanced User Experience

KDN can improve network traffic routing through efficient resource allocation and knowledge-based routing optimization, which can help improve network speed and reduce latency so that users can access network resources quickly and easily even during periods of high network traffic. Compared to SDN, KDN can have a higher level of consistent network performance as it optimizes network resources based on usage patterns, allowing users to use the network with minimal performance issues. The network reliability is much higher in KDN due to detecting and proactively correcting network issues even before they affect the users with the help of machine learning, which will help reduce network downtime and make sure users obtain continuous access to network resources. Furthermore, users feel more secure in a KDN scenario than in an SDN scenario, as the knowledge plane can detect diverse threats in real time and take corrective actions to have a minimum impact on the end user experience. Furthermore, user application performance can be improved by prioritizing network traffic based on the criticality of the application or using the QoS requirements of the specific application such that critical applications or applications with strict QoS requirements are prioritized over others. In the knowledge composition plane, user intent is considered when orchestrating rules from the composed knowledge. Thus, application plane policies are updated such that they are aligned with the high-level intents of the users and, at the same time, based on network performance. Thus, in the policy update process, high-level user intentions will not be violated, which will cause the KDN to evolve over time without violating the principle of adhering to cater the requirements of the end user.

8.1.8. Improved Network Visibility

KDN collects a wider variety of data compared to SDN. Non-conventional data, such as device sensor data, can be collected by the management plane to generate knowledge and gain more insight into the network infrastructure's availability, health, status, etc. In SDN, information-centric decisions are made based on collected information without generating knowledge or rules from it. However, knowledge has more decision-making power than raw data or information, as new patterns or concepts can be inferred from existing data. Thus, the generated knowledge provides more insight into the underlying network than information or raw data, thus providing a higher level of visibility into the network. Thus, improved visibility due to knowledge generation can affect the management and control planes to provide better network management and control decisions. KDN can provide granular network insight, including network performance at the device and application level, which helps identify any issue caused by the specific devices or applications. KDN can collect data on user behavior, including the applications that are being used and how much of each resource is consumed, which will help identify underutilization or overutilization. KDN can collect data such as traffic patterns, network device status information, network device resource utilization information, etc. to analyze patterns in the collected data using machine learning to obtain complete visibility on the network's performance, allowing it to identify areas for improvement and take corrective actions.

8.2. Challenges

As with any paradigm, we can identify challenges in knowledge-defined networks as well. The following subsections will discuss each of the identified challenges in KDN along with potential solutions for the challenges, if there are any.

8.2.1. Data

Obtaining data, in terms of both quality and quantity, is a challenge in KDN, as it relies heavily on the accuracy and completeness of the data for training machine learning models or to generate knowledge using any other model-based technique in order to self-learn and optimize network performance. In terms of quantity, for training machine learning models such as deep learning models, an enormous amount of data is required to obtain satisfactory classification or regression performance [553]. Organizations may further have difficulty collecting a large amount of data for machine learning training from real-world networks.

The data collected for training the knowledge generation models must be complete, which means that they should represent every possible state. Thus, when training, data from multiple networks under multiple network conditions must be obtained to train for every possible instance of the network. In instances where complete data cannot be obtained from a real network, the network administrators can alternatively generate synthetic data from existing data to represent unknown/new data. For instance, consider data related to DDoS attack samples, which only represent data related to a limited set of attack patterns. In order to represent unknown or new attack data, attack samples can be generated by using generative adversarial networks, and ML models can be trained on a combination of original and synthetic data representing a complete security dataset in order to have better performance. A collection of data that is incomplete, inconsistent, or biased can lead to incorrect or unreliable results, which is problematic when optimizing a complex system such as a network. Furthermore, these data must also be highly relevant to the learning task, and additional features may need to be extracted before learning. For example, in order to obtain a better classification performance for a network intrusion detection system using machine learning, specific data related to network flows, event logs, security events, etc. may need to be collected, and more pre-processing in terms of feature selection may need to be carried out to increase classification accuracy. Another requirement of the data is that they should not be outdated. However, due to the fact that networks are dynamic, data can become outdated easily. When making predictions, they should be based on the latest data that represents the state of the network. Predicting from outdated data can lead to sub-optimal performance, as the outdated data do not represent the current state of the network.

Therefore, gathering a large quantity of high-quality (relevant, complete, unbiased, consistent, and non-outdated) data is challenging, as it requires the careful attention of experts.

8.2.2. Integration with Legacy Systems

Even though programmable networking paradigms such as KDN and SDN have been proposed, many systems still rely on traditional networking systems that may not be compatible with KDN systems. KDN involves the logical decoupling of knowledge, management, and control planes from forwarding elements, whereas in traditional networks there is no knowledge plane while management and control planes are not decoupled from the data plane. These legacy networks have not been designed with the same level of flexibility and interoperability inherited by KDNs. In order for the legacy systems to become compatible with KDNs, the organizations may need to invest in making them compatible. The organizations may need to change the underlying infrastructure in order to convert their legacy networks to be compatible with KDNs. For example, data should be collected in KDN for knowledge generation. Thus, it will be difficult to convert a legacy network to send data to management and control planes, as legacy systems may not support the collection of certain types of data. The legacy networks support manual configuration and management. On the other hand, KDN relies on the automation of network configura-

tion dynamically adapting to changing network conditions, including the least amount of human intervention, which may be difficult to achieve with old legacy systems designed to be configured by humans. Furthermore, conventional networks may not support API compatibility for programmability. As the control is not logically centralized in legacy networks, they may not provide the network global level visibility and control required for the proper functioning of the KDN. Thus, a lack of visibility can act as a barrier to collecting the required data, which is essential for training or performing real-time predictions using machine learning models. Thus, because of the requirement for significant infrastructure changes, lack of flexibility, and difficulty in achieving the automation capability required for KDN, it is difficult to integrate KDN with legacy networks.

However, there are alternative proposals to integrate legacy systems with KDN without changing the legacy networks using a hybrid control approach. As discussed above, pure KDN is difficult to integrate with legacy networks. As discussed in Section 5.3.1, a hybrid KDN switch can be implemented with a local control agent that can be used for legacy network routing by exchanging control information between the forwarding elements, in addition to a KDN module that can be used for communication related to KDN. Thus, in this architecture, KDN and legacy networks can coexist.

8.2.3. Privacy

Privacy concerns can appear as the KDN paradigm involves the collection of data for network optimization by knowledge generation, which may include sensitive information of users such as user identities, confidential information of users and devices, sensitive sensor data such as private audio, video, or images, etc. There is a risk of the dissemination of such sensitive data to third parties, which can ruin the reputation of users. Thus, in KDN, private data should be handled in an ethical and secure manner. This paves the way for additional requirements for data protection measures such as encryption, access control, and monitoring to make sure that only authorized entities handle data and sensitive data are not disclosed to external parties. Furthermore, machine learning models should be trained to give fair and ethical results. If the knowledge generated by them produces a discriminatory decision, it can affect the whole organization. Thus, machine learning models should be trained with unbiased and non-discriminatory data to produce ethical results without discriminating against individuals. For example, if a machine learning model is trained in a discriminative manner to classify traffic coming from a certain group of users as lower priority, then that group of users' traffic will most likely be given low priority under any network scenario. The ethical way to train a model is without such discrimination, considering the dynamics of the network with the intention of improving network performance. Therefore, in order to mitigate privacy risk in KDN, organizations should implement robust data protection measures and ensure that knowledge generation models are trained without bias or discrimination.

8.2.4. Interoperability

At the time of writing, all control, management, knowledge, application, and data planes have well-defined standard protocols and interfaces for communication with each other, as discussed in this survey. However, different vendors may use different data formats and different machine learning or knowledge-generating models or algorithms, knowledge representation formats, rule representation formats, etc., making it difficult to integrate KDN technologies from multiple vendors into one platform. This can lead to issues with data quality, compatibility, and performance among different KDN platforms from different vendors. Furthermore, there is a lack of approaches for performance evaluation of the knowledge plane. Due to that, it will be difficult to compare the performance of KDN solutions from different networks, making it difficult to evaluate which solutions are better. For example, consider two KDN networks that detect intrusion using a supervised learning approach and an unsupervised learning approach. It is difficult to compare the classification performance of the two approaches, as they are trained from data

generated from two different KDN networks that may use different protocols, different feature selection approaches, different machine learning approaches, etc.

8.2.5. Scalability

One of the strongest scalability issues found in the KDN architecture is the amount of data collected for generating knowledge. KDN systems must obtain information from a variety of sources, including sensors, network devices, etc. The amount of data that has to be gathered and evaluated grows as a network becomes bigger. Due to the enormous quantity of data, the knowledge plane could find it difficult to develop knowledge quickly, which might cause a performance bottleneck. Machine learning models require significant computational and memory resources to run, even at low data sizes. As the size of the data increases, the machine learning algorithms may take a long time to process and may have decreased accuracy. Furthermore, KDN inherits all scalability issues found in the SDN paradigm, such as the controller scalability issue where the controller becomes overloaded in large networks, the switch scalability issue where the flow table size increases beyond the switches' memory capacities, etc.

In order to increase scalability, organizations may employ several strategies. Machine learning models such as deep learning may be used, which can process large amounts of data quickly [554]. Instead of a physically centralized knowledge plane, a physically distributed knowledge plane at the network edge may be employed to distribute the workload among multiple machine learning models whose output may be combined to generate the final output. Furthermore, management and control planes may also be implemented as a distributed approach using one of the distributed architectures discussed in Section 5.3.1 in order to improve the scalability of the KDN paradigm.

8.2.6. Reliability

The primary reliability challenge associated with KDN lies in the reliability of the data. If the data collected by sensors or devices are inaccurate or incomplete, the KDN system may make incorrect decisions, which can lead to poor network performance and increased security risks. Another reliability issue is the reliability of the machine learning or rule-based model for knowledge generation. For example, a machine learning model for traffic classification may produce subpar classification results (low accuracy, F1 score, etc.) even after being trained on comprehensive and reliable data, decreasing the system's dependability. Furthermore, data may not be able to be obtained from all the devices owing to environmental variables such as device failure, system congestion, etc. In such instances, the decisions made by the KDN systems may be inaccurate or incomplete.

To overcome these challenges, high-quality devices and sensors can be utilized in the network to ensure that correct data are received and to use machine learning algorithms trained very well for accuracy and reliability that can detect errors in data and correct themselves. Before deploying knowledge generation models in a KDN, the network administrators must make sure that the performance of the knowledge generation model is satisfactory for a test dataset in terms of performance evaluation metrics such as accuracy, F1-score, precision, etc. Furthermore, KDN systems should have additional fail-over systems to function in the case of device failures or other environmental factors.

8.2.7. Consistency

Consistency in the KDN environment can happen due to changes in the network environment, such as the addition of new devices to the system or changes to network policies. If the KDN system cannot adapt quickly to changes in the network environment, incorrect decisions may be taken by the system. If the knowledge plane is implemented as a physically distributed system and if such a system has replicas of data, data inconsistencies in replicas may lead to contradictory predictions from the machine learning models that make predictions based on such inconsistent data.

Organizations can implement change management processes to ensure that KDN systems quickly adapt to network changes. In the case of a distributed knowledge plane having replicas of data, such data should be consistent, either using eventual consistency or a strong consistency approach, to make sure that the latest version of updated data is reflected in each replicate.

8.2.8. Implementation and Maintenance Cost

KDN requires additional software, hardware, and human resources to collect, store, and analyze data using a machine learning model or another heuristic model-based method compared to SDN. The hardware and software costs are expensive to purchase and maintain, making it challenging for organizations to justify the investment. Hardware such as processing units, memory resources, additional sensors for data capture, data integration middleware, etc., and additional software such as database engines, machine learning models, knowledge-representing data structures, rule engines, rule generators, ontology editors, etc. are required. Particularly, machine learning models require high computational and memory resources to implement, which can make it challenging to purchase hardware satisfying such requirements. In order to reduce the cost of software, organizations may consider open-source options. Furthermore, KDN requires special personnel with experience in networking, data analysis, machine learning, knowledge representation models, communication protocols, languages, etc. who may provide their services for a high cost. If trained personnel are not available, they need to be educated and trained to implement and maintain KDN, which will also cost organizations.

8.2.9. Lack of Transparency

Decisions made by the control and management planes of KDN are based on knowledge generated by knowledge generation models. However, these models are difficult to be interpreted and explained. This lack of transparency can make it difficult to understand why a decision was made, especially in cases where the decisions are unexpected or seem contradictory. Furthermore, it will be challenging to identify the underlying reason for a misclassification caused by a machine learning algorithm due to a performance issue.

Using explainable machine learning approaches, which make machine learning models easier to analyze and comprehend, is one way to address the transparency problem stated above. Using explainable ML, network administrators will benefit from having a better understanding of how machine learning models operate by detecting the root causes of issues.

8.2.10. Additional Resources

In KDN, all application policy updates, network monitoring and configuration, and control plane decisions are knowledge-driven. In order to understand knowledge or rules produced from the knowledge plane, all application, management, and control planes must allocate additional resources to implement rule engines. A rule engine's purpose is to execute rules or make inferences from knowledge and make decisions based on the rule/knowledge evaluation. Rule engines infer from knowledge/rules using techniques such as forward chaining and backward chaining, which require considerable software and hardware resources for implementation. Furthermore, the whole knowledge plane, which includes knowledge generation models, knowledge bases, ontology editors, rule generators, etc., needs both software and hardware resources for knowledge or rule generation, storage, and sharing. On the other hand, in SDN, none of these additional resources are required, as it does not have a knowledge plane.

8.3. Design Guidelines

The success of a knowledge-defined network will depend on how effectively the five layers and interfaces of the KDN architecture operate. For better functioning of a KDN, all hardware, software, and human resources should be properly utilized with the collaboration of vendors, researchers, and communities. In this section, we discuss how each layer's performance can be improved with better design.

8.3.1. Switch Design

Ternary Content Addressable Memory (TCAM) is the common choice for storing a flow table in a switch. Along with TCAM, OpenFlow switches are usually designed with Static Random Access Memory (SRAM) or Field-Programmable Gate Array (FPGA) and a Graphics Processing Unit (GPU) or Central Processing Unit (CPU) [555]. OpenState has an abstraction as a superset of OpenFlow primitives to enable stateful handling of OpenFlow rules using extended finite state machines inside forwarding devices, which augment the capability and flexibility of the switches [296]. The Programming Protocol Independent Packet Processor (P4) allows the controller to specify the high-level functionality of the switches, which will be compiled into a control flow graph that can be mapped to different switches [556]. As the memory of switches is limited and may not be sufficient to store all rules when the network becomes large, several strategies, such as timeout and eviction mechanisms, flow rule aggregation, flow rule split and distribute, flow rule caching, etc., can be employed to store the rules in the limited memory [557]. In order to cope with increasing demands, such as in the case of a KDN, switches' memory, processing, and communication capacities should be upgraded. There have been attempts to improve security in switches by providing switches with an additional virtualization layer that provides virtualized security functions in the form of security applications or security service chains where the controller is responsible for activating virtualized security functions [558]. Therefore, it is better to have switches in KDN to have such virtualized security measures in the form of access control, encryption, and intrusion detection.

8.3.2. Data Collection

Data are the fundamental components that determine the performance of a knowledge-defined network. As reviewed in Sections 4 and 5, different types of data will be collected by management and control planes. Data collected by the management and control planes can be used for knowledge generation. For a given type of data, for example, network traffic flow, it is important to gather data from as many nodes as possible in order to increase the diversity and completeness of the data and reduce any bias in the dataset. As discussed earlier, adversarial training can be used to train knowledge generation models with new or unknown data. Furthermore, for the collected data, data annotation can be used to label data with additional information, such as metadata, to make them more useful for machine learning, providing a way for models to understand data more easily. Checking the obtained data for mistakes and inconsistencies is a good idea before training, as incorrect and inconsistent data might result in faulty machine learning models. Machine learning models might be trained via active learning, in which data are periodically gathered to improve the models' accuracy. Finally, information should be gathered lawfully, ethically, with the participants' consent, and with their privacy protected. Private data should be handled in an ethical and secure manner. Data protection measures such as encryption, access control, and monitoring should be employed to make sure that only authorized entities handle data, while sensitive data should not be disclosed to external parties.

8.3.3. Management Plane Design

The network management plane in KDN should be designed for dynamic network configuration, where the configuration is automated and dynamically adjusted based on real-time traffic analysis, which reduces manual intervention and speeds up the configuration process. The management plane should be designed for real-time monitoring and analytics to monitor network performance. As one of the purposes of the management plane is fault management, data should be collected by the management plane for predictive maintenance to predict potential network failures and recommend proactive measures to prevent them. KDN systems collect a vast amount of data that need to be efficiently stored and processed. Traditional data storage mechanisms may not be sufficient to handle the massive amount of data that KDN collects. Therefore, the performance and capacity of the management plane may be enhanced by intelligent data storage technologies such as distributed databases and object storage. By keeping it in memory, in-memory databases such as Redis can offer quick access to frequently used data. These databases can be used for regularly requested data such as network statistics, topology details, configuration information, etc. [559]. Time-stamped data, such as network performance measurements or telemetry data, may be stored using time-series databases such as InfluxDB, which are built for processing massive amounts of time-stamped data [560]. The management plane collects a large volume of data required for management and knowledge generation, which can be challenging to store and manage. Data compression techniques such as gzip, Snappy, etc. can be used to reduce the storage footprint of network data without compromising its quality [561]. Data deduplication techniques such as content-defined chunking or delta encoding [562] can be used to eliminate duplicate data in the management plane, as duplicate data cause excess data storage and do not effectively contribute to training machine learning models.

8.3.4. Control Plane Design

The control plane in the KDN architecture is also logically centralized, similar to the originally proposed logically centralized SDN architecture. It should be capable of handling large amounts of requests from network devices and applications, and should be able to process generated knowledge or rules from the knowledge plane. Thus, the processing capability of the controller must be very high. Controllers such as the NOX-MT use multi-threading and optimization techniques such as input–output batching to improve the controller's performance [312]. McNettle is another computationally powerful controller that can be used for KDN, as it can achieve a throughput of 14 million flows per second and reduce the number of system calls to optimize cache usage, operating system processing, and runtime system overhead [563]. In order to handle scalability issues and due to a single controller being a single point of failure, a physically distributed control architecture with multiple distributed controllers is a better design choice for the KDN. If KDN uses a set of distributed controllers with different platforms, SDN hypervisors can be used to achieve interoperability [564]. Alternatively, for distributed control systems, one may use the same controller platform to achieve maximum interoperability. For distributed controllers, state consistency using a hybrid approach, having both full consistency for critical operations that affect a large portion of the network and eventual consistency for less critical changes, is a much more desirable design approach for KDN [375].

8.3.5. Knowledge Plane Design

It is better to implement knowledge-generating models that can detect errors in data and correct themselves. It is better to use explainable machine learning instead of conventional machine learning to interpret how machine learning models make decisions and diagnose problems, in order to make knowledge generation models accountable for their decisions. The knowledge generation models must be transparent, meaning that their inner workings and decision-making processes are visible to the users. Furthermore, knowledge generation models must be robust and perform accurately and reliably under a range of

conditions and input data. The knowledge generating models must be thoroughly assessed and validated to ensure that they fulfill explainability, transparency, robustness, and other requirements. They must also be trained on a variety of representative data to guarantee that they can make appropriate judgments under a variety of circumstances. The subject, predicate, and object triplet (RDF), the ontology (OWL), the logic-based representation (KIF), or knowledge graphs (RDFS), which are organized representations of knowledge for effective querying and reasoning, can all be used to describe knowledge. A machine learning model that has been pre-trained on a big dataset and then fine-tuned on a small task-specific dataset can be improved via transfer learning. Additionally, active learning may be used to enhance the process by which knowledge-generating models are taught, allowing the models to choose the most instructive instances to utilize as training data. Additionally, attention techniques can be employed to narrowly concentrate attention on the portions of the input data that are most crucial to the activity at hand. It is required to define a clear knowledge ontology using an ontology editor for the knowledge composition plane. An ontology is a set of concepts and categories that define the relationship between different pieces of knowledge. By defining a clear knowledge ontology, different pieces of knowledge can be clearly analyzed and combined based on their relationships in the knowledge composition plane. Human expert input can be assisted in knowledge composition where possible to increase the accuracy and effectiveness of the knowledge combination process. A physically distributed knowledge plane is desired in order to distribute the workload of data among multiple knowledge generation models instead of a single model.

8.3.6. Application Plane Design

Understanding the needs of the service to be offered is crucial when designing an application. This involves determining the algorithms, data types, protocols, languages, models, etc. needed for the service's implementation. Additionally, network administrators must ensure that all applications such as traffic engineering, network management, security, settings, business, network virtualization, etc. have been deployed in order for the network to function. Even though application testing should be carried out even after application deployment to upgrade the application and discharge improved new application versions, applications ought to be extensively evaluated before deployment to ensure that they are operating correctly and meeting the service requirements. Applications should be designed to handle failures and errors, which includes implementing redundancy and fail-over mechanisms to ensure service continuity in the event of failures, irrespective of the purpose for which the application is designed. The application layer should implement appropriate security measures such as access control, authentication, encryption, etc. by implementing security applications with security policies to protect the network from unauthorized access and attacks. The APIs of a KDN should be well designed to provide developers with easy access to network functionality and knowledge. Different APIs, including RESTful-APIs, intent-based APIs, Ad Hoc APIs, Abstract APIs, etc., can be used, depending on the application type, to transmit data, knowledge/rules, intentions, and policies between the application plane and other planes. An API should have clear documentation, be simple to use, and have effective error handling. Applications should be continuously improved by collecting regular feedback from users to improve usability and the user experience. Regular updates should be released, addressing bugs and improving performance with new features. Furthermore, application personalization can be used to cater to the specific needs and preferences of individual users using various methods, such as providing recommendations based on user behavior, allowing users to customize the interface, providing contextual information, etc. Finally, it should be tested whether dynamic updating of application policies using a rule engine in the intent/policy update plane of the application plane based on the generated knowledge/rules regarding the current network status functions as desired without violating the high-level user intents.

8.3.7. Different Network Scenarios

Knowledge-Defined Optical Networks

An optical network is a telecommunications network that uses optical fibers to transmit information by converting electrical signals into optical signals that are transmitted over long distances with low signal attenuation and high bandwidth capacity, providing high-speed, reliable, and secure communication [565]. In optical networks, KDN can be specifically used to predict fiber cuts and other physical disruptions. KDN can predict if an optical fiber is likely to fail, enabling network administrators to perform maintenance before it fails. As optical networks are sensitive to changes in environmental conditions such as temperature, humidity, etc., KDN systems can be designed to account for these changes and adjust network settings accordingly. In order to achieve the above tasks, optical networks should be designed such that data from a large number of sensors on optical fibers are collected and analyzed.

Knowledge-Defined Vehicular Networks

A vehicular network is an autonomous collection of mobile devices that communicate with each other over wireless links and cooperate with each other in a distributed manner to provide network functionality in the absence of fixed infrastructure [566]. These networks are characterized by dynamic network topology, road-constrained mobility patterns, and self-organizing, dynamic, and volatile links between the wireless nodes [567]. A vehicular network uses different network communication technologies, such as Wi-Fi, cellular, and Dedicated Short Range Communication (DSRC). KDN can be designed to enable seamless handoffs between network communication technologies by using knowledge on network availability, quality, signal strength, etc. for a given application. Furthermore, KDN can be used for efficient resource allocation in vehicular networks. For instance, it can allocate more resources to safety-critical applications such as collision avoidance systems while allocating lower resources to infotainment applications. Furthermore, a knowledge-defined vehicular network can be designed to prioritize traffic received from emergency vehicles and use knowledge on congestion to optimize traffic flows.

Knowledge-Defined Internet of Things

An Internet of Things (IoT) network is an interconnected system of devices that can exchange information and communicate with one another while producing a vast amount of data that must be processed and transmitted to the cloud [568]. By analyzing patterns in data traffic and routing it to the most effective pathways, KDN may be developed with cloud computing applications to optimize the network and lower latency and network congestion. Furthermore, typical security methods such as firewalls are frequently insufficient to protect IoT devices from cyberattacks. KDN may be created to gather data from devices and produce information using a knowledge generation model to identify anomalous activity or behavior out of the ordinary in the network and to identify new dangers in order to stop them.

Knowledge-Defined Mobile Networks

A mobile network is a wireless communication system that allows mobile devices to connect to a telecommunications network and communicate with each other and other devices [569]. KDN can be designed to optimize handover procedures by analyzing data on signal strength, network congestion, etc. so that it can be handed over to the best network, ensuring a seamless transition and avoiding dropped calls.

Knowledge-Defined Wireless Sensor Networks

A wireless sensor network is a group of interconnected sensor nodes that communicate wirelessly to collect and transmit data from the physical environment [67]. KDN can be designed to predict optimal sensor placement to ensure full coverage and avoid overlap. KDN can be used to improve data collection in wireless sensor networks by analyzing network data and identifying the sensors that are producing the most accurate data. This allows for the identification and replacement of faulty sensors, leading to more reliable data collection.

8.4. Ongoing Research

There are ongoing research efforts to standardize KDNs, specifically, to introduce new protocols, languages, and interfaces for KDN to make it more interoperable.

IEEE P2302-2021 Standard for Intercloud Interoperability and Federation (SIIF) provides a reference architecture and taxonomy framework for enabling interoperability and federation between different cloud computing environments, which does not directly address knowledge-defined networking but covers a number of areas relevant to KDN [570]. This standard covers areas related to KDN such as interoperability between different cloud environments, security standards, a framework for orchestrating services across different cloud environments, the governance required for intercloud communication, and a taxonomy for describing different types of cloud services and environments. To achieve semantic interoperability between IoT frameworks, recent efforts such as IEEE P2413-2019 Standard for an Architectural Framework for the Internet of Things (IoT) have produced a report for IoT Semantic Interoperability. These efforts standardize ontologies for internet-of-things-based systems, such as domain ontologies, task ontologies, and user ontologies [571,572]. A formal explanation of the connections and concepts in a particular field is called an ontology. Semantic interoperability is the capacity of various devices and systems to interchange information and comprehend one another's meaning, even when they speak different languages or have distinct vocabularies. KDN relies heavily on ontologies because they offer a structured representation of information that can be used to analyze network activity and make choices. Diverse bits of knowledge may be integrated by creating a coherent ontology.

The Open Network Foundation (ONF) stratum project aims to enable programmability, automation, and innovation in network infrastructure, which aligns with the goals of KDN [573]. The stratum project is focused on developing an open-source, silicon-independent switch operating system that enables hardware and software disaggregation in network switches.

Metro Ethernet Forum (MEF) Lifecycle Service Orchestration (LSO) Sonata is a set of standards developed to standardize the way in which network services are orchestrated across multiple service providers and network domains. The MEF LSO Sonata attempts to standardize KDN by defining a set of APIs and data models that enable the exchange of information between different service providers and domains, which includes the use of standardized data formats such as YANG models and JSON, to represent network services and their attributes [574].

The International Telecommunication Union—Telecommunication Standardization Sector (ITU-T) study Group 13 is working on standardization for Next Generation Networks (NGNs), network virtualization of Future Networks (FNs), cloud computing, and the Internet of Things (IoT) [575]. Even though this group does not specifically focus on researching KDN, it develops standards for SDN and NFV and develops interfaces between network components in NGNs, which include KDNs.

Table 10 summarizes details of ongoing research related to the standardization of KDN.

Table 10. Ongoing research efforts to standardize KDN.

Name	Research Area	Organization	Site Address
IEEE P2302 SIIF: Reference architecture and taxonomy framework	Interoperability, security, service orchestration, governance, and taxonomy for different types of cloud services and environments	IEEE	https://standards.ieee.org/ieee/2302/7056/ (accessed on 7 April 2023)
IEEE P2413 Standard for an Architectural Framework for the Internet of Things (IoT)—Semantic interoperability	Semantic interoperability, semantic models, semantic data representation and exchange, data interoperability and integration, metadata and annotations, security and privacy, service discovery and composition, context modeling and management, ontology engineering, knowledge representation and reasoning, machine learning and data analytics, standardization, and governance of IoT	IEEE, AIOTI, oneM2M, and W3C	https://standards.ieee.org/ieee/2413/6226/ (accessed on 7 April 2023), https://standards.ieee.org/news/semantic_interoperability/ (accessed on 7 April 2023)

Table 10. *Cont.*

Name	Research Area	Organization	Site Address
Stratum project	Programmable data plane, intent-based networking, network automation, network security, network telemetry	ONF	https://opennetworking.org/stratum/ (accessed on 8 April 2023)
LSO Sonata	AI/ML network service orchestration, network slicing, intent-based networking, edge computing	MEF	https://www.mef.net/service-automation/lso-apis/inter-provider-apis/lso-sonata/ (accessed on 8 April 2023)
ITU-T group 13	Next-generation networks, future networks and emerging technologies, cloud computing	ITU-T	https://www.itu.int/en/ITU-T/about/groups/Pages/sg13.aspx (accessed on 8 April 2023)

9. Conclusions, Recommendations, and Future Research

This paper presents a comprehensive survey of knowledge-defined networking. We first presented its high-level architecture and compared KDN with SDN and traditional networking. The functions, protocols, sub-planes, models, interfaces, languages, etc. of each plane were presented with real-world examples from the existing literature. Moreover, we identified practical applications of KDN in diverse domains such as traffic engineering, network management, security, big data, cloud computing, network virtualization, and business. Among these applications, traffic engineering, network management, and security stand out as the three most dominant applications in KDN, having many applications under various sub-categories. For instance, generic application traffic engineering has routing, load-balancing, and QoS provisioning as specific applications. Furthermore, we discussed benefits, limitations, and design guidelines for KDN. Finally, we also discussed some ongoing efforts by several parties to standardize KDN in order to make it more interoperable.

As far as academic implications, this research provides a comprehensive survey and a tutorial for knowledge-defined networking. This research can be used as a very useful reference for future academicians during the innovation of new concepts related to knowledge-defined networking in different application domains. Furthermore, this work can be used to identify poorly researched or developed domains in KDN and perform more research in those areas. Additionally, this research will pave the way to point out and highlight challenges in implementing KDN and how to overcome them, which researchers can further investigate to mitigate the challenges.

Based on this survey, the following recommendations can be provided for knowledge-defined networking:

- Attention from experts is recommended to make sure KDN systems collect large quantities of high-quality data in terms of relevancy, completeness, unbiasedness, consistency, and non-outdatedness;
- In instances where legacy systems cannot be directly converted to pure KDN, a hybrid KDN approach that implements a blend of legacy and pure KDN is recommended;
- In order to secure data, knowledge, and machine learning models, robust data protection techniques such as access control, encryption, and intrusion detection systems should be used. Furthermore, for secure data, knowledge, and model sharing, KDN systems can integrate distributed secure blockchain technology;
- It is recommended to implement interoperable APIs to improve the interoperability of KDN systems that may use different standards to implement the KDN;
- Distributed knowledge generation (federated learning) and dissemination are recommended to improve the scalability of KDN systems;
- In order to improve the reliability of KDN systems, the machine learning models are recommended to be trained using high-quality data and tested thoroughly, evaluating performance evaluation metrics such as accuracy, error, F1-score, etc., before deployment in the network;

- Explainable machine learning is recommended to make sure that the ML models are accountable and explainable for the decisions they make and to aid in identifying the root causes of issues when such models make errors;
- Transfer learning and active learning are recommended to fine-tune the machine learning models' training.

Quantum machine learning is currently generating a surge of interest among researchers. It can be applied to KDNs to improve their performance, especially their computational performance, with the aid of quantum computing. By assisting quantum algorithms in KDN, this can open directions for new research and the development of innovative applications due to its performance advantage. Furthermore, future research can involve developing techniques for secure federated learning of machine learning models in a complex KDN environment.

Author Contributions: Conceptualization—P.A.D.S.N.W.; software—P.A.D.S.N.W.; validation—P.A.D.S.N.W.; formal analysis—P.A.D.S.N.W.; investigation—P.A.D.S.N.W.; resources—P.A.D.S.N.W.; writing original draft preparation—P.A.D.S.N.W.; writing review and editing—S.G.; visualization—P.A.D.S.N.W. and S.G.; supervision—S.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Acknowledgments: This research is part of a research project that principal and corresponding author Patikiri Arachchige Don Shehan Nilmantha Wijesekara is pursuing for a Degree of the University of Ruhuna.

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

References

1. Haji, S.H.; Zeebaree, S.R.; Saeed, R.H.; Ameen, S.Y.; Shukur, H.M.; Omar, N.; Sadeeq, M.A.; Ageed, Z.S.; Ibrahim, I.M.; Yasin, H.M. Comparison of software defined networking with traditional networking. *Asian J. Res. Comput. Sci.* **2021**, *9*, 1–18. [CrossRef]
2. Mishra, S.; AlShehri, M.A.R. Software defined networking: Research issues, challenges and opportunities. *Indian J. Sci. Technol.* **2017**, *10*, 1–9. [CrossRef]
3. Bhatia, J.; Modi, Y.; Tanwar, S.; Bhavsar, M. Software defined vehicular networks: A comprehensive review. *Int. J. Commun. Syst.* **2019**, *32*, e4005. [CrossRef]
4. Zhu, M.; Cai, Z.P.; Xu, M.; Cao, J.N. Software-defined vehicular networks: Opportunities and challenges. In *Energy Science and Applied Technology*; CRC Press: Boca Raton, FL, USA, 2015; pp. 247–251.
5. Nunes, B.A.A.; Mendonca, M.; Nguyen, X.N.; Obraczka, K.; Turletti, T. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 1617–1634. [CrossRef]
6. Fonseca, P.C.; Mota, E.S. A survey on fault management in software-defined networks. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2284–2321. [CrossRef]
7. Akhunzada, A.; Khan, M.K. Toward secure software defined vehicular networks: Taxonomy, requirements, and open issues. *IEEE Commun. Mag.* **2017**, *55*, 110–118. [CrossRef]
8. Zhao, L.; Li, J.; Al-Dubai, A.; Zomaya, A.Y.; Min, G.; Hawbani, A. Routing schemes in software-defined vehicular networks: Design, open issues and challenges. *IEEE Intell. Transp. Syst. Mag.* **2020**, *13*, 217–226. [CrossRef]
9. Quan, W.; Cheng, N.; Qin, M.; Zhang, H.; Chan, H.A.; Shen, X. Adaptive transmission control for software defined vehicular networks. *IEEE Wirel. Commun. Lett.* **2018**, *8*, 653–656. [CrossRef]
10. Nisar, K.; Jimson, E.R.; Hijazi, M.H.A.; Welch, I.; Hassan, R.; Aman, A.H.M.; Sodhro, A.H.; Pirbhulal, S.; Khan, S. A survey on the architecture, application, and security of software defined networking: Challenges and open issues. *Internet Things* **2020**, *12*, 100289. [CrossRef]
11. Islam, M.M.; Khan, M.T.R.; Saad, M.M.; Kim, D. Software-defined vehicular network (SDVN): A survey on architecture and routing. *J. Syst. Archit.* **2021**, *114*, 101961. [CrossRef]
12. Adbeb, T.; Wu, D.; Ibrar, M. Software-defined networking (SDN) based VANET architecture: Mitigation of traffic congestion. *Int. J. Adv. Comput. Sci. Appl.* **2020**, *11*, 706–714. [CrossRef]
13. Liu, K.; Xu, X.; Chen, M.; Liu, B.; Wu, L.; Lee, V.C. A hierarchical architecture for the future internet of vehicles. *IEEE Commun. Mag.* **2019**, *57*, 41–47. [CrossRef]
14. Toufga, S.; Abdellatif, S.; Assouane, H.T.; Owezarski, P.; Villemur, T. Towards dynamic controller placement in software defined vehicular networks. *Sensors* **2020**, *20*, 1701. [CrossRef] [PubMed]

15. Fontes, R.D.R.; Campolo, C.; Rothenberg, C.E.; Molinaro, A. From theory to experimental evaluation: Resource management in software-defined vehicular networks. *IEEE Access* **2017**, *5*, 3069–3076. [CrossRef]
16. Wang, K.; Yin, H.; Quan, W.; Min, G. Enabling collaborative edge computing for software defined vehicular networks. *IEEE Netw.* **2018**, *32*, 112–117. [CrossRef]
17. Shah, S.A.A.; Ahmed, E.; Imran, M.; Zeadally, S. 5G for vehicular communications. *IEEE Commun. Mag.* **2018**, *56*, 111–117. [CrossRef]
18. Cardona, N.; Coronado, E.; Latré, S.; Riggio, R.; Marquez-Barja, J.M. Software-defined vehicular networking: Opportunities and challenges. *IEEE Access* **2020**, *8*, 219971–219995. [CrossRef]
19. Deveaux, D.; Higuchi, T.; Uçar, S.; Härri, J.; Altintas, O. A definition and framework for vehicular knowledge networking: An application of knowledge-centric networking. *IEEE Veh. Technol. Mag.* **2021**, *16*, 57–67. [CrossRef]
20. Ucar, S.; Higuchi, T.; Wang, C.H.; Deveaux, D.; Härri, J.; Altintas, O. Vehicular knowledge networking and application to risk reasoning. In Proceedings of the Twenty-First International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, Virtual Event, 11–14 October 2020; pp. 351–356.
21. Ucar, S.; Higuchi, T.; Wang, C.H.; Deveaux, D.; Altintas, O.; Härri, J. Vehicular Knowledge Networking and Mobility-Aware Smart Knowledge Placement. In Proceedings of the 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2022; pp. 593–598.
22. Ruta, M.; Scioscia, F.; Gramegna, F.; Ieva, S.; Di Sciascio, E.; De Vera, R.P. A knowledge fusion approach for context awareness in vehicular networks. *IEEE Internet Things J.* **2018**, *5*, 2407–2419. [CrossRef]
23. Khan, M.I.; Aubet, F.X.; Pahl, M.O.; Härri, J. Deep learning-aided resource orchestration for vehicular safety communication. In Proceedings of the 2019 Wireless Days (WD), Manchester, UK, 24–26 April 2019; pp. 1–8.
24. Wijesekara, P.A.D.S.N. Deep 3D Dynamic Object Detection towards Successful and Safe Navigation for Full Autonomous Driving. *Open Transp. J.* **2022**, *16*, e187444782208191. [CrossRef]
25. Wu, D.; Li, Z.; Wang, J.; Zheng, Y.; Li, M.; Huang, Q. Vision and challenges for knowledge centric networking. *IEEE Wirel. Commun.* **2019**, *6*, 117–123. [CrossRef]
26. Liu, J.; Xu, Q. Machine learning in software defined network. In Proceedings of the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chengdu, China, 15–17 March 2019; pp. 1114–1120.
27. Clark, D.D.; Partridge, C.; Ramming, J.C.; Wroclawski, J.T. A knowledge plane for the internet. In Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Las Vegas, NV, USA, 25–29 August 2003; pp. 3–10.
28. Mestres, A.; Rodriguez-Natal, A.; Carner, J.; Barlet-Ros, P.; Alarcón, E.; Solé, M.; Muntés-Mulero, V.; Meyer, D.; Barkai, S.; Hibbett, M.J.; et al. Knowledge-defined networking. *ACM SIGCOMM Comput. Commun. Rev.* **2017**, *47*, 2–10. [CrossRef]
29. Yao, H.; Mai, T.; Xu, X.; Zhang, P.; Li, M.; Liu, Y. NetworkAI: An intelligent network architecture for self-learning control strategies in software defined networks. *IEEE Internet Things J.* **2018**, *5*, 4319–4327. [CrossRef]
30. Alzahrani, A.O.; Alenazi, M.J. Designing a network intrusion detection system based on machine learning for software defined networks. *Future Internet* **2021**, *13*, 111. [CrossRef]
31. Polat, H.; Polat, O.; Cetin, A. Detecting DDoS attacks in software-defined networks through feature selection methods and machine learning models. *Sustainability* **2020**, *12*, 1035. [CrossRef]
32. Dey, S.K.; Rahman, M.M. Effects of machine learning approach in flow-based anomaly detection on software-defined networking. *Symmetry* **2019**, *12*, 7. [CrossRef]
33. Vulpe, A.; Girla, I.; Craciunescu, R.; Berceanu, M.G. Machine learning based software-defined networking traffic classification system. In Proceedings of the 2021 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom), Bucharest, Romania, 24–28 May 2021; pp. 1–5.
34. Lim, H.K.; Kim, J.B.; Kim, K.; Hong, Y.G.; Han, Y.H. Payload-based traffic classification using multi-layer lstm in software defined networks. *Appl. Sci.* **2019**, *9*, 2550. [CrossRef]
35. Bao, K.; Matyjas, J.D.; Hu, F.; Kumar, S. Intelligent software-defined mesh networks with link-failure adaptive traffic balancing. *IEEE Trans. Cogn. Commun. Netw.* **2018**, *4*, 266–276. [CrossRef]
36. Awad, M.K.; Ahmed, M.H.H.; Almutairi, A.F.; Ahmad, I. Machine learning-based multipath routing for software defined networks. *J. Netw. Syst. Manag.* **2021**, *29*, 18. [CrossRef]
37. Toufga, S.; Abdellatif, S.; Owezarski, P.; Villemur, T.; Relizani, D. Effective prediction of V2I link lifetime and vehicle's next cell for software defined vehicular networks: A machine learning approach. In Proceedings of the 2019 IEEE Vehicular Networking Conference (VNC), Los Angeles, CA, USA, 4–6 December 2019; pp. 1–8.
38. Amari, H.; Louati, W.; Khoukhi, L.; Belguith, L.H. Securing software-defined vehicular network architecture against ddos attack. In Proceedings of the 2021 IEEE 46th Conference on Local Computer Networks (LCN), Edmonton, AB, Canada, 4–7 October 2021; pp. 653–656.
39. Zhang, D.; Yu, F.R.; Yang, R. A machine learning approach for software-defined vehicular ad hoc networks with trust management. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6.
40. Latah, M.; Toker, L. Artificial intelligence enabled software-defined networking: A comprehensive overview. *IET Netw.* **2019**, *8*, 79–99. [CrossRef]

41. Kaloxylos, A.; Gavras, A.; Camps, D.; Ghoraishi, M.; Hrasnica, H. *AI and ML—Enablers for beyond 5G Networks*; Zenodo: Geneve, Switzerland, 2021.
42. Lu, W.; Liang, L.; Kong, B.; Li, B.; Zhu, Z. AI-assisted knowledge-defined network orchestration for energy-efficient data center networks. *IEEE Commun. Mag.* **2020**, *58*, 86–92. [CrossRef]
43. Hyun, J.; Van Tu N.; Hong, J.W.K. Towards knowledge-defined networking using in-band network telemetry. In Proceedings of the NOMS 2018–2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 23–27 April 2018; pp. 1–7.
44. Hu, Y.; Li, Z.; Lan, J.; Wu, J.; Yao, L. EARS: Intelligence-driven experiential network architecture for automatic routing in software-defined networking. *China Commun.* **2020**, *17*, 149–162. [CrossRef]
45. Gosh, S.; El Boudani, B.; Dagiuklas, T.; Iqbal, M. SO-KDN: A Self-Organised Knowledge Defined Networks Architecture for Reliable Routing. In Proceedings of the 4th International Conference on Information Science and Systems, Edinburgh, UK, 17–19 March 2021; pp. 160–166.
46. Rafiq, A.; Rehman, S.; Young, R.; Song, W.C.; Khan, M.A.; Kadry, S.; Srivastava, G. Knowledge defined networks on the edge for service function chaining and reactive traffic steering. *Clust. Comput.* **2023**, *26*, 613–634. [CrossRef]
47. Duque-Torres, A.; Amezquita-Suárez, F.; Caicedo Rendon, O.M.; Ordóñez, A.; Campo, W.Y. An approach based on knowledge-defined networking for identifying heavy-hitter flows in data center networks. *Appl. Sci.* **2019**, *9*, 4808. [CrossRef]
48. Herrera, L.M.C.; Torres, A.D.; Munoz, W.Y.C. An approach based on knowledge-defined networking for identifying video streaming flows in 5G networks. *IEEE Latin Am. Trans.* **2021**, *19*, 1737–1744. [CrossRef]
49. Li, Y.; Su, X.; Ding, A.Y.; Lindgren, A.; Liu, X.; Prehofer, C.; Riekki, J.; Rahmani, R.; Tarkoma, S.; Hui, P. Enhancing the internet of things with knowledge-driven software-defined networking technology: Future perspectives. *Sensors* **2020**, *20*, 3459. [CrossRef]
50. Kreutz, D.; Ramos, F.M.; Verissimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-defined networking: A comprehensive survey. *Proc. IEEE* **2014**, *103*, 14–76. [CrossRef]
51. Zhao, Y.; Li, Y.; Zhang, X.; Geng, G.; Zhang, W.; Sun, Y. A survey of networking applications applying the software defined networking concept based on machine learning. *IEEE Access* **2019**, *7*, 95397–95417. [CrossRef]
52. Xie, J.; Yu, F.R.; Huang, T.; Xie, R.; Liu, J.; Wang, C.; Liu, Y. A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges. *IEEE Commun. Surv. Tutor.* **2018**, *21*, 393–430. [CrossRef]
53. Faezi, S.; Shirmarz, A. A Comprehensive Survey on Machine Learning using in Software Defined Networks (SDN). *Hum.-Centric Intell. Syst.* **2023**, 1–32. [CrossRef]
54. Ashtari, S.; Zhou, I.; Abolhasan, M.; Shariati, N.; Lipman, J.; Ni, W. Knowledge-defined networking: Applications, challenges and future work. *Array* **2022**, *14*, 100136. [CrossRef]
55. Safwat, M.; Elgammal, A.; AbdAllah, E.G.; Azer, M.A. Survey and taxonomy of information-centric vehicular networking security attacks. *Ad Hoc Netw.* **2022**, *124*, 102696. [CrossRef]
56. Zins, C. Conceptual approaches for defining data, information, and knowledge. *J. Am. Soc. Inf. Sci. Technol.* **2007**, *58*, 479–493. [CrossRef]
57. Trammell, B.; Casas, P.; Rossi, D.; Bär, A.; Houidi, Z.B.; Leontiadis, I.; Szemethy, T.; Mellia, M. mPlane: An intelligent measurement plane for the internet. *IEEE Commun. Mag.* **2014**, *52*, 148–156. [CrossRef]
58. Sieber, C.; Blenk, A.; Basta, A.; Hock, D.; Kellerer, W. Towards a programmable management plane for SDN and legacy networks. In Proceedings of the 2016 IEEE NetSoft Conference and Workshops (NetSoft), Seoul, Republic of Korea, 6–10 June 2016; pp. 319–327.
59. Atutxa, A.; Franco, D.; Sasiain, J.; Astorga, J.; Jacob, E. Achieving low latency communications in smart industrial networks with programmable data planes. *Sensors* **2021**, *21*, 5199. [CrossRef]
60. Hasan, K.; Ahmed, K.; Biswas, K.; Islam, M.S.; Kayes, A.S.M.; Islam, S.R. Control plane optimisation for an SDN-based WBAN framework to support healthcare applications. *Sensors* **2020**, *20*, 4200. [CrossRef]
61. Ahvar, E.; Ahvar, S.; Raza, S.M.; Manuel Sanchez Vilchez, J.; Lee, G.M. Next generation of SDN in cloud-fog for 5G and beyond-enabled applications: Opportunities and challenges. *Network* **2021**, *1*, 28–49. [CrossRef]
62. Lewis, D.; Keeney, J.; O'Sullivan, D.; Guo, S. Towards a managed extensible control plane for knowledge-based networking. In *Large Scale Management of Distributed Systems: 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM, Dublin, Ireland, 23–25 October 2006*; Proceedings 17; Springer: Berlin/Heidelberg, Germany, 2006; pp. 98–111.
63. Keeney, J.; Lewis, D.; O'Sullivan, D. Ontological semantics for distributing contextual knowledge in highly distributed autonomic systems. *J. Netw. Syst. Manag.* **2007**, *15*, 75–86. [CrossRef]
64. Jevsikova, T.; Berniukevičius, A.; Kurilovas, E. Application of resource description framework to personalise learning: Systematic review and methodology. *Inf. Educ.* **2017**, *16*, 61–82.
65. Wijesekara, P.A.D.S.N.; Wang, Y.K. A Mathematical Epidemiological Model (SEQIJRDS) to Recommend Public Health Interventions Related to COVID-19 in Sri Lanka. *COVID* **2022**, *2*, 793–826. [CrossRef]
66. Wijesekara, P.A.D.S.N. A study in University of Ruhuna for investigating prevalence, risk factors and remedies for psychiatric illnesses among students. *Sci. Rep.* **2022**, *12*, 12763. [CrossRef]
67. Seneviratne, C.; Wijesekara, P.A.D.S.N.; Leung, H. Performance analysis of distributed estimation for data fusion using a statistical approach in smart grid noisy wireless sensor networks. *Sensors* **2020**, *20*, 567. [CrossRef] [PubMed]
68. Abbasi, M.; Tahouri, R.; Rafiee, M. Enhancing the performance of the aggregated bit vector algorithm in network packet classification using GPU. *PeerJ Comput. Sci.* **2019**, *5*, e185. [CrossRef] [PubMed]

69. Liu, A.X.; Meiners, C.R.; Torng, E. Packet classification using binary content addressable memory. *IEEE/ACM Trans. Netw.* **2016**, *24*, 1295–1307. [CrossRef]
70. Rashelbach, A.; Rottenstreich, O.; Silberstein, M. A computational approach to packet classification. In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, Virtual Event, 10–14 August 2020; pp. 542–556.
71. Yang, B.; Fong, J.; Jiang, W.; Xue, Y.; Li, J. Practical multituple packet classification using dynamic discrete bit selection. *IEEE Trans. Comput.* **2012**, *63*, 424–434. [CrossRef]
72. Wang, P.C. Scalable packet classification with controlled cross-producting. *Comput. Netw.* **2009**, *53*, 821–834. [CrossRef]
73. Li, W.; Yang, T.; Rottenstreich, O.; Li, X.; Xie, G.; Li, H.; Vamanan, B.; Li, D.; Lin, H. Tuple space assisted packet classification with high performance on both search and update. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 1555–1569. [CrossRef]
74. Monks, E.M.; de Moura, B.M.P.; Schneider, G.B.; Santos, H.S.; Yamin, A.C.; Reiser, R.H.S. Towards Interval-Valued Fuzzy Approach to Video Streaming Traffic Classification. In Proceedings of the 2022 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Padua, Italy, 18–23 July 2022; pp. 1–8.
75. Jiang, J.R.; Huang, H.W.; Liao, J.H.; Chen, S.Y. Extending Dijkstra's shortest path algorithm for software defined networking. In Proceedings of the 16th Asia-Pacific Network Operations and Management Symposium, Hsinchu, Taiwan, 17–19 September 2014; pp. 1–4.
76. Hou, X.; Wu, M.; Zhao, M. An optimization routing algorithm based on segment routing in software-defined networks. *Sensors* **2018**, *19*, 49. [CrossRef]
77. Shirmarz, A.; Ghaffari, A. An adaptive greedy flow routing algorithm for performance improvement in software-defined network. *Int. J. Numer. Modell. Electron. Netw. Devices Fields* **2020**, *33*, e2676. [CrossRef]
78. Wu, J.; Qiao, X.; Nan, G. Dynamic and adaptive multi-path routing algorithm based on software-defined network. *Int. J. Distrib. Sens. Netw.* **2018**, *14*, 1550147718805689.
79. Elbasheer, M.O.; Aldegheishem, A.; Lloret, J.; Alrajeh, N. A QoS-Based routing algorithm over software defined networks. *J. Netw. Comput. Appl.* **2021**, *194*, 103215. [CrossRef]
80. Shreya, T.; Mulla, M.M.; Shinde, S.; Narayan, D.G. Ant colony Optimization-based dynamic routing in Software defined networks. In Proceedings of the 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 1–3 July 2020; pp. 1–7.
81. Assefa, B.G.; Özkasap, Ö. RESDN: A novel metric and method for energy efficient routing in software defined networks. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 736–749. [CrossRef]
82. Ha, T.; Kim, S.; An, N.; Narantuya, J.; Jeong, C.; Kim, J.; Lim, H. Suspicious traffic sampling for intrusion detection in software-defined networks. *Comput. Netw.* **2016**, *109*, 172–182. [CrossRef]
83. Dotcenko, S.; Vladyko, A.; Letenko, I. A fuzzy logic-based information security management for software-defined networks. In Proceedings of the 16th International Conference on Advanced Communication Technology, Pyeong Chang, Republic of Korea, 14–19 February 2014; pp. 167–171.
84. Yazdinejadna, A.; Parizi, R.M.; Dehghantanha, A.; Khan, M.S. A kangaroo-based intrusion detection system on software-defined networks. *Comput. Netw.* **2021**, *184*, 107688. [CrossRef]
85. Arivudainambi, D.; K.A., V.K.; Sibi Chakkaravarthy, S. LION IDS: A meta-heuristics approach to detect DDoS attacks against Software-Defined Networks. *Neural Comput. Appl.* **2019**, *31*, 1491–1501. [CrossRef]
86. Almseidin, M.; Al-Sawwa, J.; Alkasassbeh, M. Anomaly-based intrusion detection system using fuzzy logic. In Proceedings of the 2021 International Conference on Information Technology (ICIT), Amman, Jordan, 14–15 July 2021; pp. 290–295.
87. Abdullah, S.A.; Al-Hashmi, A.S. TiSEFE: Time series evolving fuzzy engine for network traffic classification. *Int. J. Commun. Netw. Inf. Secur.* **2018**, *10*, 116–124. [CrossRef]
88. Ahmadi, V.; Khorramizadeh, M. An adaptive heuristic for multi-objective controller placement in software-defined networks. *Comput. Electr. Eng.* **2018**, *66*, 204–228. [CrossRef]
89. Ros, F.J.; Ruiz, P.M. On reliable controller placements in software-defined networks. *Comput. Commun.* **2016**, *77*, 41–51. [CrossRef]
90. Hu, Y.; Luo, T.; Beaulieu, N.C.; Deng, C. The energy-aware controller placement problem in software defined networks. *IEEE Commun. Lett.* **2016**, *21*, 741–744. [CrossRef]
91. Sahoo, K.S.; Puthal, D.; Obaidat, M.S.; Sarkar, A.; Mishra, S.K.; Sahoo, B. On the placement of controllers in software-defined-WAN using meta-heuristic approach. *J. Syst. Softw.* **2018**, *145*, 180–194. [CrossRef]
92. Zelaya, C.V.G. Towards explaining the effects of data preprocessing on machine learning. In Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macao, 8–11 April 2019; pp. 2086–2090.
93. Ahmad, A.; Ostrowski, K.A.; Maślak, M.; Farooq, F.; Mehmood, I.; Nafees, A. Comparative study of supervised machine learning algorithms for predicting the compressive strength of concrete at high temperature. *Materials* **2021**, *14*, 4222. [CrossRef] [PubMed]
94. Ceriotti, M. Unsupervised machine learning in atomistic simulations, between predictions and understanding. *J. Chem. Phys.* **2019**, *150*, 150901. [CrossRef] [PubMed]
95. Van Engelen, J.E.; Hoos, H.H. A survey on semi-supervised learning. *Mach. Learn.* **2020**, *109*, 373–440. [CrossRef]
96. Comar, P.M.; Liu, L.; Saha, S.; Tan, P.N.; Nucci, A. Combining supervised and unsupervised learning for zero-day malware detection. In Proceedings of the 2013 Proceedings IEEE INFOCOM, Turin, Italy, 14–19 April 2013; pp. 2022–2030.

97. Nian, R.; Liu, J.; Huang, B. A review on reinforcement learning: Introduction and applications in industrial process control. *Comput. Chem. Eng.* **2020**, *139*, 106886. [CrossRef]
98. Herath, H.M.D.P.M.; Weraniyagoda, W.A.S.A.; Rajapaksha, R.T.M.; Wijesekara, P.A.D.S.N.; Sudheera, K.L.K.; Chong, P.H.J. Automatic Assessment of Aphasic Speech Sensed by Audio Sensors for Classification into Aphasia Severity Levels to Recommend Speech Therapies. *Sensors* **2022**, *22*, 6966. [CrossRef] [PubMed]
99. Wijesekara, P.A.D.S.N.; Gunawardena, S. A Review of Blockchain Technology in Knowledge-Defined Networking, Its Application, Benefits, and Challenges. *Network* 2023, *submitted*.
100. Wijesekara, P.A.D.S.N. An Accurate Mathematical Epidemiological Model (SEQIJRDS) to Recommend Public Health Interventions Related to COVID-19 in Sri Lanka. *Prepr. Res. Sq.* **2021**.
101. Tang, T.A.; Mhamdi, L.; McLernon, D.; Zaidi, S.A.R.; Ghogho, M.; El Moussa, F. DeepIDS: Deep learning approach for intrusion detection in software defined networking. *Electronics* **2020**, *9*, 1533. [CrossRef]
102. Li, Z.; Liu, F.; Yang, W.; Peng, S.; Zhou, J. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *33*, 6999–7019. [CrossRef]
103. Fouladi, R.F.; Ermiş, O.; Anarim, E. A novel approach for distributed denial of service defense using continuous wavelet transform and convolutional neural network for software-defined network. *Comput. Secur.* **2022**, *112*, 102524. [CrossRef]
104. Indira, B.; Valarmathi, K.; Devaraj, D. An approach to enhance packet classification performance of software-defined network using deep learning. *Soft Comput.* **2019**, *23*, 8609–8619. [CrossRef]
105. Mohammed, A.R.; Mohammed, S.A.; Shirmohammadi, S. Machine learning and deep learning based traffic classification and prediction in software defined networking. In Proceedings of the 2019 IEEE International Symposium on Measurements & Networking (M&N), Catania, Italy, 8–10 July 2019; pp. 1–6.
106. Chen, X.; Wang, X.; Yi, B.; He, Q.; Huang, M. Deep learning-based traffic prediction for energy efficiency optimization in software-defined networking. *IEEE Syst. J.* **2020**, *15*, 5583–5594. [CrossRef]
107. Dey, S.K.; Rahman, M.M. Flow based anomaly detection in software defined networking: A deep learning approach with feature selection method. In Proceedings of the 2018 4th International Conference on Electrical Engineering and Information & Communication Technology (iCEEiCT), Dhaka, Bangladesh, 13–15 September 2018; pp. 630–635.
108. Shimizu, D.Y.; Mayer, K.S.; Soares, J.A.; Arantes, D.S. A deep neural network model for link failure identification in multi-path ROADM based networks. In Proceedings of the 2020 Photonics North (PN), Niagara Falls, ON, Canada, 26–28 May 2020; p. 1.
109. Khunteta, S.; Chavva, A.K.R. Deep learning based link failure mitigation. In Proceedings of the 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, Mexico, 18–21 December 2017; pp. 806–811.
110. Reis, J.; Rocha, M.; Phan, T.K.; Griffin, D.; Le, F.; Rio, M. Deep neural networks for network routing. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; pp. 1–8.
111. Zhang, H.; Wang, T.; Liu, T.; Zhang, Q.; Liu, Y. Deep Neural Network Routing with Dynamic Space Division for 3D UAV FANETs. *Wirel. Pers. Commun.* **2022**, *125*, 2003–2028. [CrossRef]
112. Wijesekara, P.A.D.S.N.; Gunawardena, S. A Machine Learning-Aided Network Contention-Aware Link Lifetime- and Delay-Based Hybrid Routing Framework for Software-Defined Vehicular Networks. *Telecom* **2023**, *4*, 393–458. [CrossRef]
113. Azzouni, A.; Boutaba, R.; Pujolle, G. NeuRoute: Predictive dynamic routing for software-defined networks. In Proceedings of the 2017 13th International Conference on Network and Service Management (CNSM), Tokyo, Japan, 26–30 November 2017; pp. 1–6.
114. Zou, G.; Li, T.; Jiang, M.; Hu, S.; Cao, C.; Zhang, B.; Gan, Y.; Chen, Y. DeepTSQP: Temporal-aware service QoS prediction via deep neural network and feature integration. *Knowl.-Based Syst.* **2022**, *241*, 108062. [CrossRef]
115. Wu, H.; Zhang, Z.; Luo, J.; Yue, K.; Hsu, C.H. Multiple attributes QoS prediction via deep neural model with contexts. *IEEE Trans. Serv. Comput.* **2018**, *14*, 1084–1096. [CrossRef]
116. Priyanka; Kumar, D. Decision tree classifier: A detailed survey. *Int. J. Inf. Decis. Sci.* **2020**, *12*, 246–269. [CrossRef]
117. Kousar, H.; Mulla, M.M.; Shettar, P.; Narayan, D.G. Detection of DDoS attacks in software defined network using decision tree. In Proceedings of the 2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT), Bhopal, India, 24–25 April 2021; pp. 783–788.
118. Li, W.; Li, X.; Li, H.; Xie, G. Cutsplit: A decision-tree combining cutting and splitting for scalable packet classification. In Proceedings of the IEEE INFOCOM 2018—IEEE Conference on Computer Communications, Honolulu, HI, USA, 16–19 April 2018; pp. 2645–2653.
119. Liu, Z.; Sun, S.; Zhu, H.; Gao, J.; Li, J. BitCuts: A fast packet classification algorithm using bit-level cutting. *Comput. Commun.* **2017**, *109*, 38–52. [CrossRef]
120. Singh, S.; Baboescu, F.; Varghese, G.; Wang, J. Packet classification using multidimensional cutting. In Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Karlsruhe, Germany, 25–28 August 2003; pp. 213–224.
121. Ahakonye, L.A.C.; Nwakanma, C.I.; Lee, J.M.; Kim, D.S. Efficient classification of enciphered SCADA network traffic in smart factory using decision tree algorithm. *IEEE Access* **2021**, *9*, 154892–154901. [CrossRef]
122. Tong, D.; Qu, Y.R.; Prasanna, V.K. Accelerating decision tree based traffic classification on FPGA and multicore platforms. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 3046–3059. [CrossRef]

123. Yuan, Z.; Wang, C. An improved network traffic classification algorithm based on Hadoop decision tree. In Proceedings of the 2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS), Chongqing, China, 28–29 May 2016; pp. 53–56.
124. Achirul Nanda, M.; Boro Seminar, K.; Nandika, D.; Maddu, A. A comparison study of kernel functions in the support vector machine and its application for termite detection. *Information* **2018**, *9*, 5. [CrossRef]
125. Myint Oo, M.; Kamolphiwong, S.; Kamolphiwong, T.; Vasupongayya, S. Advanced support vector machine-(ASVM-) based detection for distributed denial of service (DDoS) attack on software defined networking (SDN). *J. Comput. Netw. Commun.* **2019**, *2019*, 8012568. [CrossRef]
126. Schueller, Q.; Basu, K.; Younas, M.; Patel, M.; Ball, F. A hierarchical intrusion detection system using support vector machine for SDN network in cloud data center. In Proceedings of the 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), Sydney, Australia, 21–23 November 2018; pp. 1–6.
127. Raikar, M.M.; Meena, S.M.; Mulla, M.M.; Shetti, N.S.; Karanandi, M. Data traffic classification in software defined networks (SDN) using supervised-learning. *Procedia Comput. Sci.* **2020**, *171*, 2750–2759. [CrossRef]
128. Liu, C.C.; Chang, Y.; Tseng, C.W.; Yang, Y.T.; Lai, M.S.; Chou, L.D. SVM-based classification mechanism and its application in SDN networks. In Proceedings of the 2018 10th International Conference on Communication Software and Networks (ICCSN), Chengdu, China, 6–9 July 2018; pp. 45–49.
129. Indira, B.; Valarmathi, K. A Perspective of the Machine Learning Approach for the Packet Classification in the Software Defined Network. *Intell. Autom. Soft Comput.* **2020**, *26*, 795–805. [CrossRef]
130. Fan, Z.; Liu, R. Investigation of machine learning based network traffic classification. In Proceedings of the 2017 International Symposium on Wireless Communication Systems (ISWCS), Bologna, Italy, 28–31 August 2017; pp. 1–6.
131. Speiser, J.L.; Miller, M.E.; Tooze, J.; Ip, E. A comparison of random forest variable selection methods for classification prediction modeling. *Expert Syst. Appl.* **2019**, *134*, 93–101. [CrossRef]
132. Resende, P.A.A.; Drummond, A.C. A survey of random forest based methods for intrusion detection systems. *ACM Comput. Surv.* **2018**, *51*, 1–36. [CrossRef]
133. Kokila, R.T.; Selvi, S.T.; Govindarajan, K. DDoS detection and analysis in SDN-based environment using support vector machine classifier. In Proceedings of the 2014 Sixth International Conference on Advanced Computing (ICoAC), Chennai, India, 17–19 December 2014; pp. 205–210.
134. Primartha, R.; Tama, B.A. Anomaly detection using random forest: A performance revisited. In Proceedings of the 2017 International Conference on Data and Software Engineering (ICoDSE), Palembang, Indonesia, 1–2 November 2017; pp. 1–6.
135. Amaral, P.; Dinis, J.; Pinto, P.; Bernardo, L.; Tavares, J.; Mamede, H.S. Machine learning in software defined networks: Data collection and traffic classification. In Proceedings of the 2016 IEEE 24th International Conference on Network Protocols (ICNP), Singapore, 8–11 November 2016; pp. 1–5.
136. Pasquini, R.; Stadler, R. Learning end-to-end application qos from openflow switch statistics. In Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017; pp. 1–9.
137. Fan, G.F.; Guo, Y.H.; Zheng, J.M.; Hong, W.C. Application of the weighted k-nearest neighbor algorithm for short-term load forecasting. *Energies* **2019**, *12*, 916. [CrossRef]
138. Rahman, O.; Quraishi, M.A.G.; Lung, C.H. DDoS attacks detection and mitigation in SDN using machine learning. In Proceedings of the 2019 IEEE World Congress on Services (SERVICES), Milan, Italy, 8–13 July 2019; Volume 2642, pp. 184–189.
139. Alablani, I.A.; Arafah, M.A. An SDN/ML-Based Adaptive Cell Selection Approach for HetNets: A Real-World Case Study in London, UK. *IEEE Access* **2021**, *9*, 166932–166950. [CrossRef]
140. Tuan, N.N.; Hung, P.H.; Nghia, N.D.; Tho, N.V.; Phan, T.V.; Thanh, N.H. A DDoS attack mitigation scheme in ISP networks using machine learning based on SDN. *Electronics* **2020**, *9*, 413. [CrossRef]
141. Khamaiseh, S.; Serra, E.; Li, Z.; Xu, D. Detecting saturation attacks in SDN via machine learning. In Proceedings of the 2019 4th International Conference on Computing, Communications and Security (ICCCS), Rome, Italy, 10–12 October 2019; pp. 1–8.
142. Bouacida, N.; Alghadhban, A.; Alalmaei, S.; Mohammed, H.; Shihada, B. Failure mitigation in software defined networking employing load type prediction. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–7.
143. Abar, T.; Letaifa, A.B.; El Asmi, S. Machine learning based QoE prediction in SDN networks. In Proceedings of the 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, Spain, 26–30 June 2017; pp. 1395–1400.
144. Gordon, H.; Batula, C.; Tushir, B.; Dezfouli, B.; Liu, Y. Securing smart homes via software-defined networking and low-cost traffic classification. In Proceedings of the 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), Madrid, Spain, 12–16 July 2021; pp. 1049–1057.
145. Ahmed, M.; Seraj, R.; Islam, S.M.S. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics* **2020**, *9*, 1295. [CrossRef]
146. Wang, G.; Zhao, Y.; Huang, J.; Duan, Q.; Li, J. A K-means-based network partition algorithm for controller placement in software defined network. In Proceedings of the 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 22–27 May 2016; pp. 1–6.

147. Babayiğit, B.; Banu, U.L.U. A High Available Multi-Controller Structure for SDN and Placement of Multi-Controllers of SDN with Optimized K-means Algorithm. *J. Inst. Sci. Technol.* **2021**, *11*, 2456–2466. [CrossRef]
148. Starke, A.; McNair, J.; Trevizan, R.; Bretas, A.; Peeples, J.; Zare, A. Toward resilient smart grid communications using distributed SDN with ML-based anomaly detection. In *Wired/Wireless Internet Communications: 16th IFIP WG 6.2 International Conference, WWIC 2018, Boston, MA, USA, 18–20 June 2018*; Proceedings; Springer International Publishing: Cham, Switzerland, 2018; pp. 83–94.
149. Cui, J.; Zhang, J.; He, J.; Zhong, H.; Lu, Y. DDoS detection and defense mechanism for SDN controllers with K-Means. In Proceedings of the 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), Leicester, UK, 7–10 December 2020; pp. 394–401.
150. Heigl, M.; Anand, K.A.; Urmann, A.; Fiala, D.; Schramm, M.; Hable, R. On the improvement of the isolation forest algorithm for outlier detection with streaming data. *Electronics* **2021**, *10*, 1534. [CrossRef]
151. Shahzadi, S.; Ahmad, F.; Basharat, A.; Alruwaili, M.; Alanazi, S.; Humayun, M.; Rizwan, M.; Naseem, S. Machine learning empowered security management and quality of service provision in SDN-NFV environment. *Comput. Mater. Contin.* **2021**, *66*, 2723–2749. [CrossRef]
152. Abou El Houda, Z.; Hafid, A.S.; Khoukhi, L. A novel machine learning framework for advanced attack detection using SDN. In Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 7–11 December 2021; pp. 1–6.
153. Nakahara, M.; Okui, N.; Kobayashi, Y.; Miyake, Y. Malware Detection for IoT Devices using Automatically Generated White List and Isolation Forest. In Proceedings of the IoTBDS, Online, 23–25 April 2021; pp. 38–47.
154. Zhu, J.; Jiang, M.; Liu, Z. Fault Detection and Diagnosis in Industrial Processes with Variational Autoencoder: A Comprehensive Study. *Sensors* **2021**, *22*, 227. [CrossRef]
155. Dong, S.; Su, H.; Liu, Y. A-CAVE: Network abnormal traffic detection algorithm based on variational autoencoder. *ICT Express* **2022**, 1–7. [CrossRef]
156. Bårli, E.M.; Yazidi, A.; Viedma, E.H.; Haugerud, H. DoS and DDoS mitigation using variational autoencoders. *Comput. Netw.* **2021**, *199*, 108399. [CrossRef]
157. Park, S.W.; Ko, J.S.; Huh, J.H.; Kim, J.C. Review on generative adversarial networks: focusing on computer vision and its applications. *Electronics* **2021**, *10*, 1216. [CrossRef]
158. Wang, P.; Wang, Z.; Ye, F.; Chen, X. Bytesgan: A semi-supervised generative adversarial network for encrypted traffic classification in SDN edge gateway. *Comput. Netw.* **2021**, *200*, 108535. [CrossRef]
159. Falahatraftar, F.; Pierre, S.; Chamberland, S. A Conditional Generative Adversarial Network Based Approach for Network Slicing in Heterogeneous Vehicular Networks. *Telecom* **2021**, *2*, 141–154. [CrossRef]
160. AlEroud, A.; Karabatis, G. Sdn-gan: generative adversarial deep nns for synthesizing cyber attacks on software defined networks. In *On the Move to Meaningful Internet Systems: OTM 2019 Workshops: Confederated International Workshops: EI2N, FBM, ICSP, Meta4eS and SIAnA, Rhodes, Greece, 21–25 October 2019*; Springer Nature: Cham, Switzerland, 2019; pp. 211–220.
161. Novaes, M.P.; Carvalho, L.F.; Lloret, J.; Proença, M.L., Jr. Adversarial Deep Learning approach detection and defense against DDoS attacks in SDN environments. *Future Gener. Comput. Syst.* **2021**, *125*, 156–167. [CrossRef]
162. Shi, W.; Li, Z.; Lv, W.; Wu, Y.; Chang, J.; Li, X. Laplacian support vector machine for vibration-based robotic terrain classification. *Electronics* **2020**, *9*, 513. [CrossRef]
163. Wang, P.; Lin, S.C.; Luo, M. A framework for QoS-aware traffic classification using semi-supervised machine learning in SDNs. In Proceedings of the 2016 IEEE International Conference on Services Computing (SCC), San Francisco, CA, USA, 27 June–2 July; pp. 760–765.
164. Livieris, I.E.; Kanavos, A.; Tampakas, V.; Pintelas, P. An auto-adjustable semi-supervised self-training algorithm. *Algorithms* **2018**, *11*, 139. [CrossRef]
165. Amaral, P.; Pinto, P.F.; Bernardo, L.; Mazandarani, A. Application aware SDN architecture using semi-supervised traffic classification. In Proceedings of the 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Verona, Italy, 27–29 November 2018; pp. 1–6.
166. Khamaiseh, S.; Al-Alaj, A.; Adnan, M.; Alomari, H.W. The Robustness of Detecting Known and Unknown DDoS Saturation Attacks in SDN via the Integration of Supervised and Semi-Supervised Classifiers. *Future Internet* **2022**, *14*, 164. [CrossRef]
167. Guo, X.; Bai, W. ML-SDNIDS: An attack detection mechanism for SDN based on machine learning. *Int. J. Inf. Comput. Secur.* **2022**, *19*, 118–141. [CrossRef]
168. Monshizadeh, M.; Khatri, V.; Gamdou, M.; Kantola, R.; Yan, Z. Improving data generalization with variational autoencoders for network traffic anomaly detection. *IEEE Access* **2021**, *9*, 56893–56907. [CrossRef]
169. Yang, Y.; Zheng, K.; Wu, C.; Yang, Y. Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network. *Sensors* **2019**, *19*, 2528. [CrossRef]
170. Mhamdi, L.; McLernon, D.; Zaidi, S.A.R.; Tang, T. A deep learning approach combining autoencoder with one-class SVM for DDoS attack detection in SDNs. In Proceedings of the 2020 IEEE Eighth International Conference on Communications and Networking (ComNet), Virtual Event, 28–30 October 2020; pp. 1–6.
171. Tu, J.; Ogola, W.; Xu, D.; Xie, W. Intrusion Detection Based on Generative Adversarial Network of Reinforcement Learning Strategy for Wireless Sensor Networks. *Int. J. Circuits Syst. Signal Process.* **2022**, *16*, 478–482. [CrossRef]

172. Nugraha, B.; Kulkarni, N.; Gopikrishnan, A. Detecting adversarial DDoS attacks in software-defined networking using deep learning techniques and adversarial training. In Proceedings of the 2021 IEEE International Conference on Cyber Security and Resilience (CSR), Rhodes, Greece, 26–28 July 2021; pp. 448–454.
173. Le, L.T.; Thinh, T.N. On the improvement of machine learning based intrusion detection system for SDN networks. In Proceedings of the 2021 8th NAFOSTED Conference on Information and Computer Science (NICS), Hanoi, Vietnam, 21–22 December 2021; pp. 464–469.
174. Comaneci, D.; Dobre, C. Securing networks using SDN and machine learning. In Proceedings of the 2018 IEEE International Conference on Computational Science and Engineering (CSE), Bucharest, Romania, 28–31 October 2018; pp. 194–200.
175. Da Silva, A.S.; Wickboldt, J.A.; Granville, L.Z.; Schaeffer-Filho, A. ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN. In Proceedings of the NOMS 2016—2016 IEEE/IFIP Network Operations and Management Symposium, Istanbul, Turkey, 25–29 April 2016; pp. 27–35.
176. Xu, Y.; Sun, H.; Xiang, F.; Sun, Z. Efficient DDoS detection based on K-FKNN in software defined networks. *IEEE Access* **2019**, *7*, 160536–160545. [CrossRef]
177. Feng, Y.; Cai, W.; Yue, H.; Xu, J.; Lin, Y.; Chen, J.; Hu, Z. An improved X-means and isolation forest based methodology for network traffic anomaly detection. *PLoS ONE* **2022**, *17*, e0263423. [CrossRef]
178. Deepa, V.; Sudar, K.M.; Deepalakshmi, P. Design of ensemble learning methods for DDoS detection in SDN environment. In Proceedings of the 2019 International Conference on Vision towards Emerging Trends in Communication and Networking (ViTECoN), Vellore, India, 30–31 March 201; pp. 1–6.
179. Afuwape, A.A.; Xu, Y.; Anajemba, J.H.; Srivastava, G. Performance evaluation of secured network traffic classification using a machine learning approach. *Comput. Stand. Interfaces* **2021**, *78*, 103545. [CrossRef]
180. Vithayathil Varghese, N.; Mahmoud, Q.H. A survey of multi-task deep reinforcement learning. *Electronics* **2020**, *9*, 1363. [CrossRef]
181. Swaminathan, A.; Chaba, M.; Sharma, D.K.; Ghosh, U. GraphNET: Graph neural networks for routing optimization in software defined networks. *Comput. Commun.* **2021**, *178*, 169–182. [CrossRef]
182. He, Q.; Wang, Y.; Wang, X.; Xu, W.; Li, F.; Yang, K.; Ma, L. Routing Optimization with Deep Reinforcement Learning in Knowledge Defined Networking. *IEEE Trans. Mob. Comput.* **2023**, 1–12. [CrossRef]
183. Miranda, C.; Kaddoum, G.; Boukhtouta, A.; Madi, T.; Alameddine, H.A. Intrusion Prevention Scheme Against Rank Attacks for Software-Defined Low Power IoT Networks. *IEEE Access* **2022**, *10*, 129970–129984. [CrossRef]
184. Casas-Velasco, D.M.; Rendon, O.M.C.; da Fonseca, N.L. Intelligent routing based on reinforcement learning for software-defined networking. *IEEE Trans. Netw. Serv. Manag.* **2020**, *18*, 870–881. [CrossRef]
185. Chen, Y.R.; Rezapour, A.; Tzeng, W.G.; Tsai, S.C. RL-routing: An SDN routing algorithm based on deep reinforcement learning. *IEEE Trans. Netw. Sci. Eng.* **2020**, *7*, 3185–3199. [CrossRef]
186. Guo, X.; Lin, H.; Li, Z.; Peng, M. Deep-reinforcement-learning-based QoS-aware secure routing for SDN-IoT. *IEEE Internet Things J.* **2019**, *7*, 6242–6251. [CrossRef]
187. Chen, C.; Xue, F.; Lu, Z.; Tang, Z.; Li, C. RLMR: Reinforcement Learning Based Multipath Routing for SDN. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 5124960. [CrossRef]
188. Huong, T.T.; Khoa, N.D.D.; Dung, N.X.; Thanh, N.H. A global multipath load-balanced routing algorithm based on Reinforcement Learning in SDN. In Proceedings of the 2019 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 16–18 October 2019; pp. 1336–1341.
189. Zhao, C.; Ye, M.; Xue, X.; Lv, J.; Jiang, Q.; Wang, Y. DRL-M4MR: An intelligent multicast routing approach based on DQN deep reinforcement learning in SDN. *Phys. Commun.* **2022**, *55*, 101919. [CrossRef]
190. Zhang, J.; Ye, M.; Guo, Z.; Yen, C.Y.; Chao, H.J. CFR-RL: Traffic engineering with reinforcement learning in SDN. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 2249–2259. [CrossRef]
191. Sun, P.; Guo, Z.; Lan, J.; Li, J.; Hu, Y.; Baker, T. ScaleDRL: A scalable deep reinforcement learning approach for traffic engineering in SDN with pinning control. *Comput. Netw.* **2021**, *190*, 107891. [CrossRef]
192. Akbari, I.; Tahoun, E.; Salahuddin, M.A.; Limam, N.; Boutaba, R. ATMoS: Autonomous threat mitigation in SDN using reinforcement learning. In Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–9.
193. Dake, D.K.; Gadze, J.D.; Klogo, G.S.; Nunoo-Mensah, H. Multi-agent reinforcement learning framework in sdn-iot for transient load detection and prevention. *Technologies* **2021**, *9*, 44. [CrossRef]
194. Sampaio, L.S.; Faustini, P.H.; Silva, A.S.; Granville, L.Z.; Schaeffer-Filho, A. Using NFV and reinforcement learning for anomalies detection and mitigation in SDN. In Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 25–28 June 2018; pp. 432–437.
195. Al-Jawad, A.; Comşa, I.S.; Shah, P.; Gemikonakli, O.; Trestian, R. An innovative reinforcement learning-based framework for quality of service provisioning over multimedia-based sdn environments. *IEEE Trans. Broadcast.* **2021**, *67*, 851–867. [CrossRef]
196. Xiong, Z.; Zhang, Y.; Niyato, D.; Deng, R.; Wang, P.; Wang, L.C. Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges. *IEEE Veh. Technol. Mag.* **2019**, *14*, 44–52. [CrossRef]
197. Karamplias, T.; Spantideas, S.T.; Giannopoulos, A.E.; Gkonis, P.; Kapsalis, N.; Trakadas, P. Towards Closed-Loop Automation in 5G Open RAN: Coupling an Open-Source Simulator with XApps. In Proceedings of the 2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), Grenoble, France, 7–10 June 2022; pp. 232–237.

198. Li, J.; Zhang, X. Deep reinforcement learning-based joint scheduling of eMBB and URLLC in 5G networks. *IEEE Wirel. Commun. Lett.* **2020**, *9*, 1543–1546. [CrossRef]
199. Sapra, D.; Pimentel, A.D. Deep learning model reuse and composition in knowledge centric networking. In Proceedings of the 2020 29th International Conference on Computer Communications and Networks (ICCCN), Honolulu, HI, USA, 3–8 August 2020; pp. 1–11.
200. Wijesekara, P.A.D.S.N.; Sudheera, K.L.K.; Sandamali, G.G.N.; Chong, P.H.J. Machine Learning Based Link Stability Prediction for Routing in Software Defined Vehicular Networks. In Proceedings of the 20th Academic Sessions, Matara, Sri Lanka, 7 June 2023; p. 60.
201. Tudorache, T.; Nyulas, C.; Noy, N.F.; Musen, M.A. WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the web. *Semant. Web* **2013**, *4*, 89–99. [CrossRef] [PubMed]
202. Futia, G.; Vetrò, A. On the integration of knowledge graphs into deep learning models for a more comprehensible AI—Three challenges for future research. *Information* **2020**, *11*, 122. [CrossRef]
203. McBride, B. The resource description framework (RDF) and its vocabulary description language RDFS. In *Handbook on Ontologies*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 51–65.
204. Antoniou, G.; Harmelen, F.V. Web ontology language: Owl. In *Handbook on Ontologies*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 91–110.
205. Fensel, D.; Van Harmelen, F.; Horrocks, I.; McGuinness, D.L.; Patel-Schneider, P.F. OIL: An ontology infrastructure for the semantic web. *IEEE Intell. Syst.* **2001**, *16*, 38–45. [CrossRef]
206. Yang, Y.; Calmet, J. Ontobayes: An ontology-driven uncertainty model. In Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), Vienna, Austria, 28–30 November 2005; Volume 1, pp. 457–463.
207. Hayes, P.; Menzel, C. A semantics for the knowledge interchange format. In Proceedings of the IJCAI 2001 Workshop on the IEEE Standard Upper Ontology, Seattle, WA, USA, 4 August 2001; Volume 145, p. 145.
208. Jarvis, M.P.; Nuzzo-Jones, G.; Heffernan, N.T. Applying machine learning techniques to rule generation in intelligent tutoring systems. In *Intelligent Tutoring Systems: 7th International Conference, ITS, Maceió, Brazil, 30 August–3 September 2004*; Proceedings 7; Springer: Berlin/Heidelberg, Germany, 2004; pp. 541–553.
209. Boley, H.; Tabet, S.; Wagner, G. Design rationale for RuleML: A markup language for semantic web rules. In Proceedings of the SWWS, Stanford, CA, USA, 30 July–1 August 2001; Volume 1, pp. 381–401.
210. Horrocks, I.; Patel-Schneider, P.F.; Boley, H.; Tabet, S.; Grosof, B.; Dean, M. SWRL: A semantic web rule language combining OWL and RuleML. *W3C Memb. Submiss.* **2004**, *21*, 1–31.
211. Kifer, M. Rule interchange format: The framework. In *Web Reasoning and Rule Systems: Second International Conference, RR 2008, Karlsruhe, Germany, 31 October–1 November 2008*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 1–11.
212. Inoue, T.; Mano, T.; Mizutani, K.; Minato, S.I.; Akashi, O. Rethinking packet classification for global network view of software-defined networking. In Proceedings of the 2014 IEEE 22nd International Conference on Network Protocols, Raleigh, NC, USA, 21–24 October 2014; pp. 296–307.
213. Li, C.; Li, T.; Li, J.; Shi, Z.; Wang, B. Enabling packet classification with low update latency for SDN switch on FPGA. *Sustainability* **2020**, *12*, 3068. [CrossRef]
214. Wang, P.; Ye, F.; Chen, X.; Qian, Y. Datanet: Deep learning based encrypted network traffic classification in sdn home gateway. *IEEE Access* **2018**, *6*, 55380–55391. [CrossRef]
215. Malik, A.; de Fréin, R.; Al-Zeyadi, M.; Andreu-Perez, J. Intelligent SDN traffic classification using deep learning: Deep-SDN. In Proceedings of the 2020 2nd International Conference on Computer Communication and the Internet (ICCCI), Nagoya, Japan, 26–29 June 2020; pp. 184–189.
216. Chen, X.F.; Yu, S.Z. CIPA: A collaborative intrusion prevention architecture for programmable network and SDN. *Comput. Secur.* **2016**, *58*, 1–19. [CrossRef]
217. Pratama, R.F.; Suwastika, N.A.; Nugroho, M.A. Design and implementation adaptive Intrusion Prevention System (IPS) for attack prevention in software-defined network (SDN) architecture. In Proceedings of the 2018 6th International Conference on Information and Communication Technology (ICoICT), Bandung, Indonesia, 3–5 May 2018; pp. 299–304.
218. Siregar, B.; Purba, R.F.D.; Fahmi, F. Intrusion Prevention System Against Denial of Service Attacks Using Genetic Algorithm. In Proceedings of the 2018 IEEE International Conference on Communication, Networks and Satellite (Comnetsat), Medan, Indonesia, 15–17 November 2018; pp. 55–59.
219. Girdler, T.; Vassilakis, V.G. Implementing an intrusion detection and prevention system using Software-Defined Networking: Defending against ARP spoofing attacks and Blacklisted MAC Addresses. *Comput. Electr. Eng.* **2021**, *90*, 106990. [CrossRef]
220. Bera, S.; Misra, S.; Jamalipour, A. FlowStat: Adaptive flow-rule placement for per-flow statistics in SDN. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 530–539. [CrossRef]
221. Yahyaoui, H.; Aidi, S.; Zhani, M.F. On using flow classification to optimize traffic routing in SDN networks. In Proceedings of the 2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 January 2020; pp. 1–6.
222. Xu, S.; Wang, X.; Yang, G.; Ren, J.; Wang, S. Routing optimization for cloud services in SDN-based Internet of Things with TCAM capacity constraint. *J. Commun. Netw.* **2020**, *22*, 145–158. [CrossRef]

223. Wang, M.; Liu, J.; Mao, J.; Cheng, H.; Chen, J.; Qi, C. RouteGuardian: Constructing secure routing paths in software-defined networking. *Tsinghua Sci. Technol.* **2017**, *22*, 400–412. [CrossRef]
224. Yan, B.; Liu, Q.; Shen, J.; Liang, D. Flowlet-level multipath routing based on graph neural network in OpenFlow-based SDN. *Future Gener. Comput. Syst.* **2022**, *134*, 140–153. [CrossRef]
225. Saha, N.; Misra, S.; Bera, S. Q-flag: QoS-aware flow-rule aggregation in software-defined IoT networks. *IEEE Internet Things J.* **2021**, *9*, 4899–4906. [CrossRef]
226. Mondal, A.; Misra, S.; Maity, I. AMOPE: Performance analysis of OpenFlow systems in software-defined networks. *IEEE Syst. J.* **2019**, *14*, 124–131. [CrossRef]
227. Guo, Y.; Hu, G.; Shao, D. QOGMP: QoS-oriented global multi-path traffic scheduling algorithm in software defined network. *Sci. Rep.* **2022**, *12*, 14600. [CrossRef]
228. Alishahi, M.; Yaghmaee Moghaddam, M.H.; Pourreza, H.R. Multi-class routing protocol using virtualization and SDN-enabled architecture for smart grid. *Peer-to-Peer Netw. Appl.* **2018**, *11*, 380–396. [CrossRef]
229. Kim, W.; Sharma, P.; Lee, J.; Banerjee, S.; Tourrilhes, J.; Lee, S.J.; Yalagandula, P. Automated and Scalable QoS Control for Network Convergence. In Proceedings of the INM/WREN'10: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, San Jose, CA, USA, 27 April 2010; Volume 10, p. 1.
230. Bagci, K.T.; Tekalp, A.M. SDN-enabled distributed open exchange: Dynamic QoS-path optimization in multi-operator services. *Comput. Netw.* **2019**, *162*, 106845. [CrossRef]
231. Khan, S.; Hussain, F.K.; Hussain, O.K. Guaranteeing end-to-end QoS provisioning in SOA based SDN architecture: A survey and Open Issues. *Future Gener. Comput. Syst.* **2021**, *119*, 176–187. [CrossRef]
232. Cao, B.; Sun, Z.; Zhang, J.; Gu, Y. Resource allocation in 5G IoV architecture based on SDN and fog-cloud computing. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 3832–3840. [CrossRef]
233. Liu, D.; Gu, T.; Xue, J.P. Rule engine based on improvement rete algorithm. In Proceedings of the 2010 International Conference on Apperceiving Computing and Intelligence Analysis Proceeding, Chengdu, China, 17–19 December 2010; pp. 346–349.
234. Proctor, M. Drools: A rule engine for complex event processing. In *Applications of Graph Transformations with Industrial Relevance: 4th International Symposium, AGTIVE 2011, Budapest, Hungary, 4–7 October 2011*; Revised Selected and Invited Papers 4; Springer: Berlin/Heidelberg, Germany, 2022; p. 2.
235. Carral, D.; Dragoste, I.; González, L.; Jacobs, C.; Krötzsch, M.; Urbani, J. Vlog: A rule engine for knowledge graphs. In *The Semantic Web—ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, 26–30 October 2019*; Proceedings, Part II 18; Springer International Publishing: Berlin/Heidelberg, Germany, 2019; pp. 19–35.
236. Jang, M.; Sohn, J.C. Bossam: An extended rule engine for OWL inferencing. In *Rules and Rule Markup Languages for the Semantic Web: Third International Workshop, RuleML, Hiroshima, Japan, 8 November 2004*; Proceedings 3; Springer: Berlin/Heidelberg, Germany, 2004; pp. 128–138.
237. Wygant, R.M. CLIPS—A powerful development and delivery expert system tool. *Comput. Ind. Eng.* **1989**, *7*, 546–549. [CrossRef]
238. Friedman-Hill, E. *Jess, the Rule Engine for the Java Platform*; Sandia National Laboratories: Albuquerque, NM, USA, 2008.
239. Barbieri, D.F.; Braga, D.; Ceri, S.; Valle, E.D.; Grossniklaus, M. C-SPARQL: A continuous query language for RDF data streams. *Int. J. Semant. Comput.* **2010**, *4*, 3–25. [CrossRef]
240. O'Connor, M.J.; Das, A.K. SQWRL: A query language for OWL. *OWLED* **2009**, *529*, 1–8.
241. Liu, P.; Wang, X.; Fu, Q.; Yang, Y.; Li, Y.F.; Zhang, Q. KGVQL: A knowledge graph visual query language with bidirectional transformations. *Knowl.-Based Syst.* **2022**, *250*, 108870. [CrossRef]
242. Finin, T.; McKay, D.P.; Fritzson, R.; McEntire, R. KQML: An information and knowledge exchange protocol. In *Knowledge Building and Knowledge Sharing*; IOS Press: Amsterdam, The Netherlands, 1994.
243. Taelman, R.; Vander Sande, M.; Verborgh, R. GraphQL-LD: Linked data querying with GraphQL. In Proceedings of the ISWC2018, the 17th International Semantic Web Conference, Monterey, CA, USA, 8–12 October 2018; pp. 1–4.
244. Bjorklund, M. *YANG—A Data Modeling Language for the Network Configuration Protocol (NETCONF)*; No. rfc6020; Internet Engineering Task Force: Fremont, CA, USA, 2010.
245. Uslar, M.; Specht, M.; Rohjans, S.; Trefke, J.; González, J.M. *The Common Information Model CIM: IEC 61968/61970 and 62325—A Practical Introduction to the CIM*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012.
246. Sieber, C.; Blenk, A.; Hock, D.; Scheib, M.; Höhn, T.; Köhler, S.; Kellerer, W. Network configuration with quality of service abstractions for SDN and legacy networks. In Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, 11–15 May 2015; IEEE: Toulouse, France, 2015; p. 11351136.
247. Xu, C.; Li, P.; Luo, Y. A programmable policy engine to facilitate time-efficient science DMZ management. *Future Gener. Comput. Syst.* **2018**, *89*, 515–524. [CrossRef]
248. Tuncer, D.; Charalambides, M.; Clayman, S.; Pavlou, G. Adaptive resource management and control in software defined networks. *IEEE Trans. Netw. Serv. Manag.* **2015**, *12*, 18–33. [CrossRef]
249. Ballani, H.; Francis, P. Conman: A step towards network manageability. *ACM SIGCOMM Comput. Commun. Rev.* **2007**, *37*, 205–216. [CrossRef]
250. Chen, X.; Mao, Z.M.; Van der Merwe, J. PACMAN: A platform for automated and controlled network operations and configuration management. In Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, Rome, Italy, 1 December 2009; pp. 277–288.

251. Narisetty, R.; Dane, L.; Malishevskiy, A.; Gurkan, D.; Bailey, S.; Narayan, S.; Mysore, S. OpenFlow configuration protocol: Implementation for the of management plane. In Proceedings of the 2013 Second GENI Research and Educational Experiment Workshop, Salt Lake, UT, USA, 20–22 March 2013; IEEE: Toulouse, France, 2013; pp. 66–67.
252. Martinez, A.; Yannuzzi, M.; de Vergara, J.L.; Serral-Gracià, R.; Ramírez, W. An ontology-based information extraction system for bridging the configuration gap in hybrid sdn environments. In Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, 11–15 May 2015; IEEE: Toulouse, France, 2015; pp. 441–449.
253. Lu, H.; Arora, N.; Zhang, H.; Lumezanu, C.; Rhee, J.; Jiang, G. Hybnet: Network manager for a hybrid network infrastructure. In Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference, Beijing, China, 11–13 December 2013; pp. 1–6.
254. Safrianti, E.; Sari, L.O.; Sari, N.A. Real-Time Network Device Monitoring System with Simple Network Management Protocol (SNMP) Model. In Proceedings of the 2021 3rd International Conference on Research and Academic Community Services (ICRACOS), Virtual Event, 9–10 October 2021; IEEE: Toulouse, France, 2021; pp. 122–127.
255. Chowdhury, S.R.; Bari, M.F.; Ahmed, R.; Boutaba, R. Payless: A low cost network monitoring framework for software defined networks. In Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; IEEE: Toulouse, France, 2014.
256. Suh, J.; Kwon, T.T.; Dixon, C.; Felter, W.; Carter, J. Opensample: A low-latency, sampling-based measurement platform for commodity sdn. In Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems, Madrid, Spain, 30 June–3 July 2014; IEEE: Toulouse, France, 2014; pp. 228–237.
257. Van Adrichem, N.L.; Doerr, C.; Kuipers, F.A. Opennetmon: Network monitoring in openflow software-defined networks. In Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; IEEE: Toulouse, France, 2014.
258. Sun, P.; Yu, M.; Freedman, M.J.; Rexford, J.; Walker, D. Hone: Joint host-network traffic management in software-defined networks. *J. Netw. Syst. Manag.* **2015**, *23*, 374–399. [CrossRef]
259. Fan, Y.; Zhang, N. A survey on software-defined vehicular networks. *J. Comput.* **2017**, *28*, 236–244.
260. Wijesekara, P.A.D.S.N.; Sudheera, K.L.K.; Sandamali, G.G.N.; Chong, P.H.J. An Optimization Framework for Data Collection in Software Defined Vehicular Networks. *Sensors* **2023**, *23*, 1600. [CrossRef] [PubMed]
261. Wette, P.; Karl, H. Which flows are hiding behind my wildcard rule? Adding packet sampling to OpenFlow. In Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, Hong Kong, China, 12–16 August 2013; ACM: New York, NY, USA, 2013; pp. 541–542.
262. Zhou, D.; Yan, Z.; Liu, G.; Atiquzzaman, M. An adaptive network data collection system in sdn. *IEEE Trans. Cogn. Commun. Netw.* **2019**, *6*, 562–574. [CrossRef]
263. Liao, W.H.; Kuai, S.C. An energy-efficient sdn-based data collection strategy for wireless sensor networks. In Proceedings of the 2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2), Kanazawa, Japan, 22–25 November 2017; IEEE: Toulouse, France, 2017; pp. 91–97.
264. Jiao, Z.; Ding, H.; Dang, M.; Tian, R.; Zhang, B. Predictive big data collection in vehicular networks: A software defined networking based approach. In Proceedings of the 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, USA, 4–8 December 2016; IEEE: Toulouse, France, 2016; pp. 1–6.
265. Wang, R.; Gu, C.; He, S.; Shi, Z.; Meng, W. An interoperable and flat Industrial Internet of Things architecture for low latency data collection in manufacturing systems. *J. Syst. Archit.* **2022**, *129*, 102631. [CrossRef]
266. Sugadev, M.; Rayen, S.J.; Harirajkumar, J.; Rathi, R.; Anitha, G.; Ramesh, S.; Ramaswamy, K. Implementation of Combined Machine Learning with the Big Data Model in IoMT Systems for the Prediction of Network Resource Consumption and Improving the Data Delivery. *Comput. Intell. Neurosci.* **2022**, *2022*, 6510934. [CrossRef]
267. Tian, Y.; Chen, W.; Lea, C.T. An SDN-based traffic matrix estimation framework. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 1435–1445. [CrossRef]
268. Hadem, P.; Saikia, D.K.; Moulik, S. An SDN-based intrusion detection system using SVM with selective logging for IP traceback. *Comput. Netw.* **2021**, *191*, 108015. [CrossRef]
269. Tarnaras, G.; Haleplidis, E.; Denazis, S. SDN and ForCES based optimal network topology discovery. In Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), London, UK, 13–17 April 2015; IEEE: Toulouse, France, 2015; pp. 1–6.
270. Khalid, A.; Rehman, R.A.; Burhan, M. CBILEM: A novel energy aware mobility handling protocol for SDN based NDN-MANETs. *Ad Hoc Netw.* **2023**, *140*, 103049. [CrossRef]
271. Trivisonno, R.; Guerzoni, R.; Vaishnavi, I.; Frimpong, A. Network resource management and QoS in SDN-enabled 5G systems. In Proceedings of the 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, USA, 6–10 December 2015; IEEE: Toulouse, France, 2015; pp. 1–7.
272. Sezer, S.; Scott-Hayward, S.; Chouhan, P.K.; Fraser, B.; Lake, D.; Finnegan, J.; Viljoen, N.; Miller, M.; Rao, N. Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Commun. Mag.* **2013**, *51*, 36–43. [CrossRef]
273. Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N.; Shenker, S. NOX: Towards an operating system for networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 105–110. [CrossRef]
274. Song, P.; Liu, Y.; Liu, C.; Qian, D. ParaFlow: Fine-grained parallel SDN controller for large-scale networks. *J. Netw. Comput. Appl.* **2017**, *87*, 46–59. [CrossRef]

275. Shin, S.; Song, Y.; Lee, T.; Lee, S.; Chung, J.; Porras, P.; Yegneswaran, V.; Noh, J.; Kang, B.B. Rosemary: A robust, secure, and high-performance network operating system. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014; ACM: New York, NY, USA, 2014; pp. 78–89.
276. Li, L.; Chou, W.; Zhou, W.; Luo, M. Design patterns and extensibility of REST API for networking applications. *IEEE Trans. Netw. Serv. Manag.* **2016**, *13*, 154–167. [CrossRef]
277. Cajas, C.D.; Budanov, D.O. SDN applications and plugins in the opendaylight controller. In Proceedings of the 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg and Moscow, Russia, 27–30 January 2020; IEEE: Toulouse, France, 2020; pp. 9–13.
278. Rowshanrad, S.; Abdi, V.; Keshtgari, M. Performance evaluation of SDN controllers: Floodlight and OpenDaylight. *IIUM Eng. J.* **2016**, *17*, 47–57. [CrossRef]
279. Uddin, R.; Monir, M.F. Performance analysis of SDN based firewalls: POX vs. ODL. In Proceedings of the 2019 5th International Conference on Advances in Electrical Engineering (ICAEE), Dhaka, Bangladesh, 26–28 September 2019; IEEE: Toulouse, France, 2019; pp. 691–698.
280. Sanvito, D.; Moro, D.; Gulli, M.; Filippini, I.; Capone, A.; Campanella, A. ONOS Intent Monitor and Reroute service: Enabling plug&play routing logic. In Proceedings of the 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), Montreal, QC, Canada, 25–29 June 2018; IEEE: Toulouse, France, 2018; pp. 272–276.
281. Tuncer, D.; Charalambides, M.; Tangari, G.; Pavlou, G. A northbound interface for software-based networks. In Proceedings of the 2018 14th International Conference on Network and Service Management (CNSM), Rome, Italy, 5–9 November 2018; IEEE: Toulouse, France, 2018; pp. 99–107.
282. Casey, C.J.; Sutton, A.; Sprintson, A. tinyNBI: Distilling an API from essential OpenFlow abstractions. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 22 August 2014; pp. 37–42.
283. Hinrichs, T.L.; Gude, N.S.; Casado, M.; Mitchell, J.C.; Shenker, S. Practical declarative network management. In Proceedings of the 1st ACM workshop on Research on Enterprise Networking, Barcelona, Spain, 21 August 2009; ACM: New York, NY, USA, 2009; pp. 1–10.
284. Voellmy, A.; Agarwal, A.; Hudak, P. *Nettle: Functional Reactive Programming for Openflow Networks*; Yale University New Haven Ct Department of Computer Science: New Haven, CT, USA, 2010.
285. Voellmy, A.; Kim, H.; Feamster, N. Procera: A language for high-level reactive network control. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks, Helsinki, Finland, 13 August 2012; pp. 43–48.
286. Foster, N.; Freedman, M.J.; Harrison, R.; Rexford, J.; Meola, M.L.; Walker, D. Frenetic: A high-level language for OpenFlow networks. In Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow, Philadelphia, PA, USA, 30 November 2010; ACM: New York, NY, USA, 2010; pp. 1–6.
287. Kim, H.; Reich, J.; Gupta, A.; Shahbaz, M.; Feamster, N.; Clark, R. Kinetic: Verifiable dynamic network control. In Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI, Oakland, CA, USA, 4–6 May 2015; pp. 59–72.
288. Batista, B.L.A.; Fernandez, M.P. Ponderflow: A new policy specification language to sdn openflow-based networks. *Int. J. Adv. Netw. Serv.* **2014**, *7*, 163–172.
289. Voellmy, A.; Wang, J.; Yang, Y.R.; Ford, B.; Hudak, P. Maple: Simplifying SDN programming using algorithmic policies. *ACM SIGCOMM Comput. Commun. Rev.* **2013**, *43*, 87–98. [CrossRef]
290. Soulé, R.; Basu, S.; Marandi, P.J.; Pedone, F.; Kleinberg, R.; Sirer, E.G.; Foster, N. Merlin: A language for managing network resources. *IEEE/ACM Trans. Netw.* **2018**, *26*, 2188–2201. [CrossRef]
291. Shalimov, A.; Zuikov, D.; Zimarina, D.; Pashkov, V.; Smeliansky, R. Advanced study of SDN/OpenFlow controllers. In Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, Moscow, Russia, 24–25 October 2013; ACM: New York, NY, USA, 2013; pp. 1–6.
292. Haleplidis, E.; Salim, J.H.; Halpern, J.M.; Hares, S.; Pentikousis, K.; Ogawa, K.; Wang, W.; Denazis, S.; Koufopavlou, O. Network programmability with ForCES. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 1423–1440. [CrossRef]
293. Paliwal, M.; Shrimankar, D.; Tembhurne, O. Controllers in SDN: A review report. *IEEE Access* **2018**, *6*, 36256–36270. [CrossRef]
294. Li, S.; Hu, D.; Fang, W.; Ma, S.; Chen, C.; Huang, H.; Zhu, Z. Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability. *IEEE Netw.* **2017**, *31*, 58–66. [CrossRef]
295. Eadala, S.Y.; Nagarajan, V. A review on deployment architectures of path computation element using software defined networking paradigm. *Indian J. Sci. Technol.* **2016**, *9*, 1–10. [CrossRef]
296. Bianchi, G.; Bonola, M.; Capone, A.; Cascone, C. Openstate: Programming platform-independent stateful openflow applications inside the switch. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 44–51. [CrossRef]
297. Rotsos, C.; King, D.; Farshad, A.; Bird, J.; Fawcett, L.; Georgalas, N.; Gunkel, M.; Shiomoto, K.; Wang, A.; Mauthe, A.; et al. Network service orchestration standardization: A technology survey. *Comput. Stand. Interfaces* **2017**, *54*, 203–215. [CrossRef]
298. Bannour, F.; Souihi, S.; Mellouk, A. Distributed SDN control: Survey, taxonomy, and challenges. *IEEE Commun. Surv. Tutor.* **2017**, *20*, 333–354. [CrossRef]
299. Berde, P.; Gerola, M.; Hart, J.; Higuchi, Y.; Kobayashi, M.; Koide, T.; Lantz, B.; O'Connor, B.; Radoslavov, P.; Snow, W.; et al. ONOS: Towards an open, distributed SDN OS. In Proceedings of the third workshop on Hot topics in software defined networking, Chicago, IL, USA, 22 August 2014; ACM: New York, NY, USA, 2014; pp. 1–6.

300. Koponen, T.; Casado, M.; Gude, N.; Stribling, J.; Poutievski, L.; Zhu, M.; Ramanathan, R.; Iwata, Y.; Inoue, H.; Hama, T.; et al. Onix: A distributed control platform for large-scale production networks. *OSDI* **2010**, *10*, 6.
301. Kurniawan, M.T.; Moszardo, I.; Almaarif, A. Network Slicing On Software Defined Network Using Flowvisor and POX Controller To Flowspace Isolation Enforcement. In Proceedings of the 2022 10th International Conference on Smart Grid (icSmartGrid), Istanbul, Turkey, 27–29 June 2022; pp. 29–34.
302. Nurkahfi, G.N.; Mitayani, A.; Mardiana, V.A.; Dinata, M.M.M. Comparing flowvisor and open virtex as SDN-based site-to-site VPN services solution. In Proceedings of the 2019 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET), Tangerang, Indonesia, 23–24 October 2019; IEEE: Toulouse, France, 2019; pp. 142–147.
303. Drutskoy, D.A. Software-Defined Network Virtualization with FlowN. Ph.D. Thesis, Princeton University, Princeton, NJ, USA, 2012.
304. Blenk, A.; Basta, A.; Kellerer, W. HyperFlex: An SDN virtualization architecture with flexible hypervisor function allocation. In Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, 11–15 May 2015; IEEE: Toulouse, France, 2015; pp. 397–405.
305. Yoo, Y.; Yang, G.; Lee, J.; Shin, C.; Kim, H.; Yoo, C. TeaVisor: Network Hypervisor for Bandwidth Isolation in SDN-NV. *IEEE Trans. Cloud Comput.* **2022**, 1–17. [CrossRef]
306. Ijari, P. Comparison between Cisco ACI and VMWARE NSX. *IOSR J. Comput. Eng. (IOSR-JCE)* **2017**, *19*, 70–72. [CrossRef]
307. Tello, A.M.D.; Abolhasan, M. SDN controllers scalability and performance study. In Proceedings of the 2019 13th International conference on signal processing and communication systems (ICSPCS), Gold Coast, QLD, 16–18 December 2019; pp. 1–10.
308. Hassas Yeganeh, S.; Ganjali, Y. Kandoo: A framework for efficient and scalable offloading of control applications. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks, Helsinki, Finland, 13 August 2012; ACM: New York, NY, USA, 2012; pp. 19–24.
309. Santos, M.A.; Nunes, B.A.; Obraczka, K.; Turletti, T.; De Oliveira, B.T.; Margi, C.B. Decentralizing SDN's control plane. In Proceedings of the 39th Annual IEEE Conference on Local Computer Networks, Edmonton, AB, Canada, 8–11 September 2014; IEEE: Toulouse, France, 2014; pp. 402–405.
310. Curtis, A.R.; Mogul, J.C.; Tourrilhes, J.; Yalagandula, P.; Sharma, P.; Banerjee, S. DevoFlow: Scaling flow management for high-performance networks. In Proceedings of the ACM SIGCOMM 2011 Conference, Toronto, ON, Canada, 15–19 August 2011; ACM: New York, NY, USA, 2011; pp. 254–265.
311. Ku, I.; Lu, Y.; Gerla, M.; Gomes, R.L.; Ongaro, F.; Cerqueira, E. Towards software-defined VANET: Architecture and services. In Proceedings of the 2014 13th annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET), Piran, Slovenia, 2–4 June 2014; IEEE: Toulouse, France, 2014; pp. 103–110.
312. Salman, O.; Elhajj, I.H.; Kayssi, A.; Chehab, A. SDN controllers: A comparative study. In Proceedings of the 2016 18th Mediterranean Electrotechnical Conference (MELECON), Lemesos, Cyprus, 18–20 April 2016; IEEE: Toulouse, France, 2016; pp. 1–6.
313. Erickson, D. The beacon openflow controller. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, China, 16 August 2015; ACM: New York, NY, USA, 2013; pp. 13–18.
314. Al-Alaj, A.; Sandhu, R.; Krishnan, R. A formal access control model for se-floodlight controller. In Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, Richardson, TX, USA, 27 March 2019; ACM: New York, NY, USA, 2019; pp. 1–6.
315. Vahlenkamp, M.; Schneider, F.; Kutscher, D.; Seedorf, J. Enabling information centric networking in IP networks using SDN. In Proceedings of the 2013 IEEE SDN for Future Networks and Services (SDN4FNS), Trento, Italy, 11–13 November 2013; IEEE: Toulouse, France, 2013; pp. 1–6.
316. Sheikh, M.N.A.; Halder, M. SDN-Based approach to evaluate the best controller: Internal controller NOX and external controllers POX, ONOS, RYU. *Glob. J. Comput. Sci. Technol.* **2019**, *19*, 21–32. [CrossRef]
317. Banikazemi, M.; Olshefski, D.; Shaikh, A.; Tracey, J.; Wang, G. Meridian: An SDN platform for cloud network services. *IEEE Commun. Mag.* **2013**, *51*, 120–127. [CrossRef]
318. Vladyko, A.; Muthanna, A.; Kirichek, R. Comprehensive SDN testing based on model network. In *Internet of Things, Smart Spaces, and Next Generation Networks and Systems: 16th International Conference, NEW2AN 2016, and 9th Conference, ruSMART, St. Petersburg, Russia, 26–28 September 2016*; Proceedings 16; Springer International Publishing: Berlin/Heidelberg, Germany, 2016; pp. 539–549.
319. Botelho, F.; Bessani, A.; Ramos, F.M.; Ferreira, P. On the design of practical fault-tolerant SDN controllers. In Proceedings of the 2014 Third European Workshop on Software Defined Networks, Budapest, Hungary, 1–3 September 2014; IEEE: Toulouse, France, 2014; pp. 73–78.
320. Katta, N.; Zhang, H.; Freedman, M.; Rexford, J. Ravana: Controller fault-tolerance in software-defined networking. In Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, Santa Clara, CA, USA, 17–18 June 2015; ACM: New York, NY, USA, 2015; pp. 1–12.
321. Cowen, L.J. Compact routing with minimum stretch. *J. Algorithms* **2001**, *38*, 170–183. [CrossRef]
322. Fu, Y.; Bi, J.; Gao, K.; Chen, Z.; Wu, J.; Hao, B. Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks. In Proceedings of the 2014 IEEE 22nd International Conference on Network Protocols, Raleigh, NC, USA, 21–24 October 2014; IEEE: Toulouse, France, 2014; pp. 569–576.

323. Marconett, D.; Yoo, S.B. Flowbroker: A software-defined network controller architecture for multi-domain brokering and reputation. *J. Netw. Syst. Manag.* **2015**, *23*, 328–359. [CrossRef]
324. Jain, S.; Kumar, A.; Mandal, S.; Ong, J.; Poutievski, L.; Singh, A.; Venkata, S.; Wanderer, J.; Zhou, J.; Zhu, M.; et al. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Comput. Commun. Rev.* **2013**, *43*, 3–14. [CrossRef]
325. Yap, K.K.; Motiwala, M.; Rahe, J.; Padgett, S.; Holliman, M.; Baldus, G.; Hines, M.; Kim, T.; Narayanan, A.; Jain, A.; et al. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, Los Angeles, CA, USA, 21–25 August 2017; ACM: New York, NY, USA, 2017; pp. 432–445.
326. McCauley, J.; Panda, A.; Casado, M.; Koponen, T.; Shenker, S. Extending SDN to large-scale networks. *Open Netw. Summit* **2013**, 1–2.
327. Schriegel, S.; Kobzan, T.; Jasperneite, J. Investigation on a distributed SDN control plane architecture for heterogeneous time sensitive networks. In Proceedings of the 2018 14th IEEE International Workshop on Factory Communication Systems (WFCS), Imperia, Italy, 13–15 June 2018; IEEE: Toulouse, France, 2018; pp. 1–10.
328. Phemius, K.; Bouet, M.; Leguay, J. DISCO: Distributed SDN controllers in a multi-domain environment. In Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; IEEE: Toulouse, France, 2014.
329. Gupta, A.; Vanbever, L.; Shahbaz, M.; Donovan, S.P.; Schlinker, B.; Feamster, N.; Rexford, J.; Shenker, S.; Clark, R.; Katz-Bassett, E. Sdx: A software defined internet exchange. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 551–562. [CrossRef]
330. Stringer, J.; Pemberton, D.; Fu, Q.; Lorier, C.; Nelson, R.; Bailey, J.; Corrêa, C.N.; Rothenberg, C.E. Cardigan: SDN distributed routing fabric going live at an Internet exchange. In Proceedings of the 2014 IEEE Symposium on Computers and Communications (ISCC), Funchal, Portugal, 23–26 June 2014; IEEE: Toulouse, France, 2014; pp. 1–7.
331. Chung, J.; Cox, J.; Ibarra, J.; Bezerra, J.; Morgan, H.; Clark, R.; Owen, H. AtlanticWave-SDX: An international SDX to support science data applications. In Proceedings of the Software Defined Networking (SDN) for Scientific Networking Workshop, Online, 20 November 2015; Volume 15, pp. 1–7.
332. Dey, P.K.; Yuksel, M. Hybrid cloud integration of routing control & data planes. In Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking, Irvine, CA, USA, 12 December 2016; ACM: New York, NY, USA, 2016; pp. 25–30.
333. Wang, W.; He, W.; Su, J. Enhancing the effectiveness of traffic engineering in hybrid SDN. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; IEEE: Toulouse, France, 2017; pp. 1–6.
334. Yu, M.; Rexford, J.; Freedman, M.J.; Wang, J. Scalable flow-based networking with DIFANE. *ACM SIGCOMM Comput. Commun. Rev.* **2010**, *40*, 351–362. [CrossRef]
335. Feng, T.; Bi, J. OpenRouteFlow: Enable legacy router as a software-defined routing service for hybrid SDN. In Proceedings of the 2015 24th International Conference on Computer Communication and Networks (ICCCN), Las Vegas, NV, USA, 3–6 August 2015; IEEE: Toulouse, France, 2015; pp. 1–8.
336. Vissicchio, S.; Vanbever, L.; Rexford, J. Sweet little lies: Fake topologies for flexible routing. In Proceedings of the 13th ACM Workshop on Hot Topics in Networks, Los Angeles, CA, USA, 27–28 October 2014; ACM: New York, NY, USA, 2014; pp. 1–7.
337. Kaur, S.; Singh, J.; Ghumman, N.S. Network programmability using POX controller. In *ICCCS International Conference on Communication, Computing & Systems*; IEEE: Toulouse, France, 2014; Volume 138, p. 70.
338. Huang, S.; Zhao, J.; Wang, X. HybridFlow: A lightweight control plane for hybrid SDN in enterprise networks. In Proceedings of the 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), Beijing, China, 20–21 June 2016; IEEE: Toulouse, France, 2016; pp. 1–2.
339. Kandoi, R.; Antikainen, M. Denial-of-service attacks in OpenFlow SDN networks. In Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, 11–15 May 2015; IEEE: Toulouse, France, 2015; pp. 1322–1326.
340. Gonzalez, C.; Flauzac, O.; Nolot, F.; Jara, A. A novel distributed SDN-secured architecture for the IoT. In Proceedings of the 2016 International Conference on Distributed Computing in Sensor Systems (DCOSS), Washington, DC, USA, 26–28 May 2016; IEEE: Toulouse, France, 2016; pp. 244–249.
341. Nguyen, T.G.; Phan, T.V.; Hoang, D.T.; Nguyen, T.N.; So-In, C. Efficient SDN-based traffic monitoring in IoT networks with double deep Q-network. In Proceedings of the Computational Data and Social Networks: 9th International Conference, CSoNet 2020, Dallas, TX, USA, 11–13 December 2020; Springer International Publishing: Cham, Switzerland, 2021; pp. 26–38.
342. Li, F.; Cao, J.; Wang, X.; Sun, Y.; Pan, T.; Liu, X. Adopting SDN switch buffer: Benefits analysis and mechanism design. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 2171–2176.
343. Zhang, C.; Hu, G.; Chen, G.; Sangaiah, A.K.; Zhang, P.A.; Yan, X.; Jiang, W. Towards a SDN-based integrated architecture for mitigating IP spoofing attack. *IEEE Access* **2017**, *6*, 22764–22777. [CrossRef]
344. Achleitner, S.; La Porta, T.; Jaeger, T.; McDaniel, P. Adversarial network forensics in software defined networking. In Proceedings of the Symposium on SDN Research, Santa Clara, CA, USA, 3–4 April 2017; pp. 8–20.
345. Jin, C.; Lumezanu, C.; Xu, Q.; Mekky, H.; Zhang, Z.L.; Jiang, G. Magneto: Unified fine-grained path control in legacy and openflow hybrid networks. In Proceedings of the Symposium on SDN Research, Santa Clara, CA, USA, 3–4 April 2017; pp. 75–87.

346. Lorenz, C.; Hock, D.; Scherer, J.; Durner, R.; Kellerer, W.; Gebert, S.; Gray, N.; Zinner, T.; Tran-Gia, P. An SDN/NFV-enabled enterprise network architecture offering fine-grained security policy enforcement. *IEEE Commun. Mag.* **2017**, *55*, 217–223. [CrossRef]
347. Wang, Y.; Bi, J.; Lin, P.; Lin, Y.; Zhang, K. SDI: A multi-domain SDN mechanism for fine-grained inter-domain routing. *Ann. Telecommun.* **2016**, *71*, 625–637. [CrossRef]
348. Huang, J.; He, Y.; Duan, Q.; Yang, Q.; Wang, W. Admission control with flow aggregation for QoS provisioning in software-defined network. In Proceedings of the 2014 IEEE Global Communications Conference, Austin, TX, USA, 8–12 December 2014; pp. 1182–1186.
349. Zhang, X.; Cheng, Z.; Lin, R.; He, L.; Yu, S.; Luo, H. Local fast reroute with flow aggregation in software defined networks. *IEEE Commun. Lett.* **2016**, *21*, 785–788. [CrossRef]
350. Lu, Y.; Fu, B.; Xi, X.; Zhang, Z.; Wu, H. An SDN-based flow control mechanism for guaranteeing QoS and maximizing throughput. *Wirel. Pers. Commun.* **2017**, *97*, 417–442. [CrossRef]
351. Ajaeiya, G.A.; Adalian, N.; Elhajj, I.H.; Kayssi, A.; Chehab, A. Flow-based intrusion detection system for SDN. In Proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC), Heraklion, Greece, 3–6 July 2017; pp. 787–793.
352. Huang, N.; Li, Q.; Lin, D.; Lit, X.; Shen, G.; Jiang, Y. Software-defined label switching: Scalable per-flow control in SDN. In Proceedings of the 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), Banff, AB, Canada, 4–6 June 2018; pp. 1–10.
353. Kosugiyama, T.; Tanabe, K.; Nakayama, H.; Hayashi, T.; Yamaoka, K. A flow aggregation method based on end-to-end delay in SDN. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–6.
354. Dridi, L.; Zhani, M.F. SDN-guard: DoS attacks mitigation in SDN networks. In Proceedings of the 2016 5th IEEE International Conference on Cloud Networking (Cloudnet), Pisa, Italy, 3–5 October 2016; pp. 212–217.
355. Malboubi, M.; Wang, L.; Chuah, C.N.; Sharma, P. Intelligent SDN based traffic (de) aggregation and measurement paradigm (iSTAMP). In Proceedings of the IEEE INFOCOM 2014—IEEE Conference on Computer Communications, Toronto, ON, Canada, 27 April–2 May 2014; pp. 934–942.
356. Hyder, M.F.; Ismail, M.A. Securing control and data planes from reconnaissance attacks using distributed shadow controllers, reactive and proactive approaches. *IEEE Access* **2021**, *9*, 21881–21894. [CrossRef]
357. Bianco, A.; Giaccone, P.; Mashayekhi, R.; Ullio, M.; Vercellone, V. Scalability of ONOS reactive forwarding applications in ISP networks. *Comput. Commun.* **2017**, *102*, 130–138. [CrossRef]
358. Dusi, M.; Bifulco, R.; Gringoli, F.; Schneider, F. Reactive logic in software-defined networking: Measuring flow-table requirements. In Proceedings of the 2014 International Wireless Communications and Mobile Computing Conference (IWCMC), Nicosia, Cyprus, 4–8 August 2014; pp. 340–345.
359. Petroulakis, N.E.; Fysarakis, K.; Askoxylakis, I.; Spanoudakis, G. Reactive security for SDN/NFV-enabled industrial networks leveraging service function chaining. *Trans. Emerg. Telecommun. Technol.* **2018**, *29*, e3269. [CrossRef]
360. Zerkane, S.; Espes, D.; Le Parc, P.; Cuppens, F. Software defined networking reactive stateful firewall. In *ICT Systems Security and Privacy Protection: 31st IFIP TC 11 International Conference, SEC 2016, Ghent, Belgium, 30 May–1 June 2016*; Proceedings 31; Springer International Publishing: Cham, Switzerland, 2016; pp. 119-132.
361. Hamza, A.; Gharakheili, H.H.; Sivaraman, V. Combining MUD policies with SDN for IoT intrusion detection. In Proceedings of the 2018 Workshop on IoT Security and Privacy, Budapest, Hungary, 20 August 2018; pp. 1–7.
362. Singh, M.P.; Bhandari, A. New-flow based DDoS attacks in SDN: Taxonomy, rationales, and research challenges. *Comput. Commun.* **2020**, *154*, 509–527. [CrossRef]
363. Thakur, A.K.S.; Rawat, S. Utility Based Framework for Reactive and Proactive Congestion Control in SDN. In Proceedings of the 2021 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), Hyderabad, India, 13–16 December 2021; pp. 396–401.
364. Braun, W.; Menth, M. Software-defined networking using OpenFlow: Protocols, applications and architectural design choices. *Future Internet* **2014**, *6*, 302–336. [CrossRef]
365. Abou El Houda, Z.; Khoukhi, L.; Hafid, A. Chainsecure-a scalable and proactive solution for protecting blockchain applications using sdn. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6.
366. Azab, M.; Fortes, J.A. Towards proactive SDN-controller attack and failure resilience. In Proceedings of the 2017 International Conference on Computing, Networking and Communications (ICNC), Santa Clara, CA, USA, 26–29 January 2017; pp. 442–448.
367. Zhou, Y.; Cheng, G.; Yu, S. An SDN-enabled proactive defense framework for DDoS mitigation in IoT networks. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 5366–5380. [CrossRef]
368. Shakeri, S.; Parsaeefard, S.; Derakhshani, M. Proactive admission control and dynamic resource management in SDN-based virtualized networks. In Proceedings of the 2017 8th International Conference on the Network of the Future (NOF), London, UK, 22–24 November 2017; pp. 46–51.
369. Nadar, S.; Chaudhari, S. Proactive-routing path update in Software Defined Networks (SDN). In Proceedings of the 2017 International Conference on Intelligent Computing and Control (I2C2), Coimbatore, India, 23–24 June 2017; pp. 1–3.

370. Padma, V.; Yogesh, P. Proactive failure recovery in OpenFlow based software defined networks. In Proceedings of the 2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN), Chennai, India, 26–28 March 2015; pp. 1–6.
371. Aslan, M.; Matrawy, A. Adaptive consistency for distributed SDN controllers. In Proceedings of the 2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks), Warsaw, Poland, 27–30 September 2016; pp. 150–157.
372. Semong, T.; Maupong, T.; Anokye, S.; Kehulakae, K.; Dimakatso, S.; Boipelo, G.; Sarefo, S. Intelligent load balancing techniques in software defined networks: A survey. *Electronics* **2020**, *9*, 1091. [CrossRef]
373. Hanmer, R.; Jagadeesan, L.; Mendiratta, V.; Zhang, H. Friend or foe: Strong consistency vs. overload in high-availability distributed systems and SDN. In Proceedings of the 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Memphis, TN, USA, 15–18 October 2018; pp. 59–64.
374. Bannour, F.; Souihi, S.; Mellouk, A. Adaptive state consistency for distributed onos controllers. In Proceedings of the 2018 IEEE Global Communications Conference (Globecom), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–7.
375. Sakic, E.; Sardis, F.; Guck, J.W.; Kellerer, W. Towards adaptive state consistency in distributed SDN control plane. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–7.
376. Ahmad, S.; Mir, A.H. Scalability, consistency, reliability and security in SDN controllers: A survey of diverse SDN controllers. *J. Netw. Syst. Manag.* **2021**, *29*, 9. [CrossRef]
377. Nougnanke, K.B.; Bruyere, M.; Labit, Y. Low-overhead near-real-time flow statistics collection in SDN. In Proceedings of the 2020 6th IEEE Conference on Network Softwarization (NetSoft), Ghent, Belgium, 29 June–3 July 2020; pp. 155–159.
378. Miguel, C.J.; Neto, F.J.; Santos, J.A.; Sampaio, P.N. Data collection in sdn networks with contextual analysis. In Proceedings of the 2019 14th Iberian Conference on Information Systems and Technologies (CISTI), Coimbra, Portugal, 19–22 June 2019; pp. 1–6.
379. Lin, H.; Yan, Z.; Chen, Y.; Zhang, L. A survey on network security-related data collection technologies. *IEEE Access* **2018**, *6*, 18345–18365. [CrossRef]
380. EL-Garoui, L.; Pierre, S.; Chamberland, S. A new SDN-based routing protocol for improving delay in smart city environments. *Smart Cities* **2020**, *3*, 1004–1021. [CrossRef]
381. Zhu, M.; Cao, J.; Pang, D.; He, Z.; Xu, M. SDN-based routing for efficient message propagation in VANET. In *Wireless Algorithms, Systems, and Applications: 10th International Conference, WASA 2015, Qufu, China, 10–12 August 2015*; Proceedings 10; Springer International Publishing: Cham, Switzerland, 2015; pp. 788–797.
382. Amokrane, A.; Langar, R.; Boutaba, R.; Pujolle, G. Flow-based management for energy efficient campus networks. *IEEE Trans. Netw. Serv. Manag.* **2015**, *12*, 565–579. [CrossRef]
383. Dutra, D.L.C.; Bagaa, M.; Taleb, T.; Samdanis, K. Ensuring end-to-end QoS based on multi-paths routing using SDN technology. In Proceedings of the GLOBECOM 2017—2017 IEEE Global Communications Conference, Singapore, 4–8 December 2017; pp. 1–6.
384. Muthumanikandan, V.; Valliyammai, C. Link failure recovery using shortest path fast rerouting technique in SDN. *Wirel. Pers. Commun.* **2017**, *97*, 2475–2495. [CrossRef]
385. Moghaddam, F.F.; Wieder, P.; Yahyapour, R. Policy Engine as a Service (PEaaS): an approach to a reliable policy management framework in cloud computing environments. In Proceedings of the 2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud), Vienna, Austria, 22–24 August 2016; pp. 137–144.
386. Polčák, L.; Caldarola, L.; Choukir, A.; Cuda, D.; Dondero, M.; Ficara, D.; Franková, B.; Holkovič, M.; Muccifora, R.; Trifilo, A. High level policies in SDN. In *E-Business and Telecommunications: 12th International Joint Conference, ICETE 2015, Colmar, France, 20–22 July 2015*; Revised Selected Papers 12; Springer International Publishing: Cham, Switzerland, 2016; pp. 39–57.
387. Maldonado-Lopez, F.A.; Calle, E.; Donoso, Y. Detection and prevention of firewall-rule conflicts on software-defined networking. In Proceedings of the 2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM), Munich, Germany, 5–7 October 2015; pp. 259–265.
388. Bari, M.F.; Chowdhury, S.R.; Ahmed, R.; Boutaba, R. PolicyCop: An autonomic QoS policy enforcement framework for software defined networks. In Proceedings of the 2013 IEEE SDN for Future Networks and Services (SDN4FNS), Trento, Italy, 11–13 November 2013; pp. 1–7.
389. Sevilla, M.A.; Maltzahn, C.; Alvaro, P.; Nasirigerdeh, R.; Settlemyer, B.W.; Perez, D.; Rich, D.; Shipman, G.M. Programmable caches with a data management language and policy engine. In Proceedings of the 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Washington, DC, USA, 1–4 May 2018; pp. 203–212.
390. Kao, M.T.; Huang, B.X.; Kao, S.J.; Tseng, H.W. An effective routing mechanism for link congestion avoidance in software-defined networking. In Proceedings of the 2016 International Computer Symposium (ICS), Chiayi, Taiwan, 15–17 December 2016; pp. 154–158.
391. Zehra, U.; Shah, M.A. A survey on resource allocation in software defined networks (SDN). In Proceedings of the 2017 23rd International Conference on Automation and Computing (ICAC), Huddersfield, UK, 7–8 September 2017; pp. 1–6.
392. Henni, D.E.; Ghomari, A.; Hadjadj-Aoul, Y. A consistent QoS routing strategy for video streaming services in SDN networks. *Int. J. Communic. Syst.* **2020**, *33*, e4177. [CrossRef]
393. Capone, A.; Cascone, C.; Nguyen, A.Q.; Sanso, B. Detour planning for fast and reliable failure recovery in SDN with OpenState. In Proceedings of the 2015 11th International Conference on the Design of Reliable Communication Networks (DRCN), Vilanova, Spain, 28–31 March 2015; pp. 25–32.

394. Reich, J.; Monsanto, C.; Foster, N.; Rexford, J.; Walker, D. *Modular SDN Programming with Pyretic*; Technical Reprot of USENIX; USENIX: Berkeley, CA, USA, 2013; 30p.
395. Yang, G.; Jin, H.; Kang, M.; Moon, G.J.; Yoo, C. Network monitoring for SDN virtual networks. In Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications, Virtual, 6–9 July 2020; pp. 1261–1270.
396. Li, Y.; Chen, M. Software-defined network function virtualization: A survey. *IEEE Access* **2015**, *3*, 2542–2553.
397. Chen, Y.J.; Wang, L.C.; Lin, F.Y.; Lin, B.S.P. Deterministic quality of service guarantee for dynamic service chaining in software defined networking. *IEEE Trans. Netw. Serv. Manag.* **2017**, *14*, 991–1002. [CrossRef]
398. Martini, B.; Paganelli, F.; Mohammed, A.A.; Gharbaoui, M.; Sgambelluri, A.; Castoldi, P. SDN controller for context-aware data delivery in dynamic service chaining. In Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), London, UK, 13–17 April 2015; pp. 1–5.
399. Da Costa Cordeiro, W.L.; Marques, J.A.; Gaspary, L.P. Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management. *J. Netw. Syst. Manag.* **2017**, *25*, 784–818. [CrossRef]
400. Tseng, J.; Wang, R.; Tsai, J.; Edupuganti, S.; Min, A.W.; Woo, S.; Junkins, S.; Tai, T.Y.C. Exploiting integrated GPUs for network packet processing workloads. In Proceedings of the 2016 IEEE NetSoft Conference and Workshops (NetSoft), Seoul, Republic of Korea, 6–10 June 2016; pp. 161–165.
401. Dane, L.; Gurkan, D. GENI with a network processing unit: enriching SDN application experiments. In Proceedings of the 2014 Third GENI Research and Educational Experiment Workshop, Atlanta, GA, USA, 19–20 March 2014; pp. 9–14.
402. Han, S.; Jang, K.; Park, K.; Moon, S. PacketShader: a GPU-accelerated software router. *ACM SIGCOMM Comput. Commun. Rev.* **2010**, *40*, 195–206. [CrossRef]
403. Yazdinejad, A.; Bohlooli, A.; Jamshidi, K. Performance improvement and hardware implementation of open flow switch using FPGA. In Proceedings of the 2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI), Tehran, Iran, 28 February–1 March 2019; pp. 515–520.
404. Khattak, M.K.; Tang, Y.; Khan, U.S. TOSwitch: Programmable and high-throughput switch using hybrid switching chips. *IEEE Commun. Lett.* **2019**, *23*, 2266–2270. [CrossRef]
405. Lu, G.; Miao, R.; Xiong, Y.; Guo, C. Using cpu as a traffic co-processing unit in commodity switches. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks, Helsinki, Finland, 13 August 2012; pp. 31–36.
406. Csikor, L.; Szalay, M.; Rétvári, G.; Pongrácz, G.; Pezaros, D.P.; Toka, L. Transition to SDN is HARMLESS: Hybrid architecture for migrating legacy ethernet switches to SDN. *IEEE/ACM Trans. Netw.* **2020**, *28*, 275–288. [CrossRef]
407. Rahimi, R.; Veeraraghavan, M.; Nakajima, Y.; Takahashi, H.; Okamoto, S.; Yamanaka, N. A high-performance OpenFlow software switch. In Proceedings of the 2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR), Yokohama, Japan, 14–17 June 2016; pp. 93–99.
408. Chen, T.S.; Lee, D.Y.; Liu, T.T.; Wu, A.Y. Dynamic reconfigurable ternary content addressable memory for OpenFlow-compliant low-power packet processing. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2016**, *63*, 1661–1672. [CrossRef]
409. Lee, D.Y.; Wang, C.C.; Wu, A.Y. Bundle-updatable SRAM-based TCAM design for openflow-compliant packet processor. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, **27**, 1450–1454. [CrossRef]
410. Zhu, H.; Fan, H.; Luo, X.; Jin, Y. Intelligent timeout master: Dynamic timeout for SDN-based data centers. In Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, 11–15 May 2015; pp. 734–737.
411. Luo, S.; Yu, H.; Li, L. Practical flow table aggregation in SDN. *Comput. Netw.* **2015**, *92*, 72–88. [CrossRef]
412. Dong, M.; Li, H.; Ota, K.; Xiao, J. Rule caching in SDN-enabled mobile access networks. *IEEE Netw.* **2015**, *29*, 40–45. [CrossRef]
413. Kanizo, Y.; Hay, D.; Keslassy, I. Palette: Distributing tables in software-defined networks. In Proceedings of the 2013 Proceedings IEEE INFOCOM, Turin, Italy, 14–19 April 2013; pp. 545–549.
414. Tanyingyong, V.; Hidell, M.; Sjödin, P. Improving pc-based openflow switching performance. In Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, La Jolla, CA, USA, 25–26 October 2010; pp. 1–2.
415. Tanyingyong, V.; Hidell, M.; Sjödin, P. Using hardware classification to improve pc-based openflow switching. In Proceedings of the 2011 IEEE 12th International Conference on High Performance Switching and Routing, Cartagena, Spain, 4–6 July 2011; pp. 215–221.
416. Emmerich, P.; Raumer, D.; Wohlfart, F.; Carle, G. Assessing soft- and hardware bottlenecks in PC-based packet forwarding systems. In Proceedings of the ICN 2015—The Fourteenth International Conference on Networks, Barcelona, Spain, 19–24 April 2015; Volume 90.
417. Lockwood, J.W.; McKeown, N.; Watson, G.; Gibb, G.; Hartke, P.; Naous, J.; Raghuraman, R.; Luo, J. NetFPGA—An open platform for gigabit-rate network switching and routing. In Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education (MSE'07), San Diego, CA, USA, 3–4 June 2007; pp. 160–161.
418. Ghaffar, Z.; Alshahrani, A.; Fayaz, M.; Alghamdi, A.M.; Gwak, J. A topical review on machine learning, software defined networking, internet of things applications: Research limitations and challenges. *Electronics* **2021**, *10*, 880.
419. Barrett, R.; Facey, A.; Nxumalo, W.; Rogers, J.; Vatcher, P.; St-Hilaire, M. Dynamic traffic diversion in SDN: Testbed vs. mininet. In Proceedings of the 2017 International Conference on Computing, Networking and Communications (ICNC), Santa Clara, CA, USA, 26–29 January 2017; pp. 167–171.

420. Tu, W.; Wei, Y.H.; Antichi, G.; Pfaff, B. Revisiting the open vSwitch dataplane ten years later. In Proceedings of the 2021 ACM SIGCOMM 2021 Conference, Virtual Event, 23–27 August 2021; pp. 245–257.
421. Poutievski, L.; Mashayekhi, O.; Ong, J.; Singh, A.; Tariq, M.; Wang, R.; Zhang, J.; Beauregard, V.; Conner, P.; Gribble, S.; et al. Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking. In Proceedings of the ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, 22–26 August 2022; pp. 66–85.
422. Das, S.; Parulkar, G.; McKeown, N.; Singh, P.; Getachew, D.; Ong, L. Packet and circuit network convergence with OpenFlow. In Proceedings of the 2010 Conference on Optical Fiber Communication (OFC/NFOEC), Collocated National Fiber Optic Engineers Conference, San Diego, CA, USA, 21–25 March 2010; pp. 1–3.
423. Azodolmolky, S.; Nejabati, R.; Escalona, E.; Jayakumar, R.; Efstathiou, N.; Simeonidou, D. Integrated OpenFlow—GMPLS control plane: An overlay model for software defined packet over optical networks. In Proceedings of the 2011 37th European Conference and Exhibition on Optical Communication, Geneva, Switzerland, 18–22 September 2011; pp. 1–3.
424. Dizdarević, S.; Dizdarević, H.; Škrbić, M.; Hadžiahmetović, N. A survey on transition from GMPLS control plane for optical multilayer networks to SDN control plane. In Proceedings of the 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 30 May–3 June 2016; pp. 537–544.
425. Xue, X.; Nakamura, F.; Prifti, K.; Pan, B.; Yan, F.; Wang, F.; Guo, X.; Tsuda, H.; Calabretta, N. SDN enabled flexible optical data center network with dynamic bandwidth allocation based on photonic integrated wavelength selective switch. *Opt. Express* **2020**, *28*, 8949–8958. [CrossRef] [PubMed]
426. Khan, I.; Tunesi, L.; Masood, M.U.; Ghillino, E.; Bardella, P.; Carena, A.; Curri, V. Automatic management of N × N photonic switch powered by machine learning in software-defined optical transport. *IEEE Open J. Commun. Soc.* **2021**, *2*, 1358–1365. [CrossRef]
427. Wijesekara, P.A.D.S.N.; Sangeeth, W.M.A.K.; Perera, H.S.C.; Jayasundere, N.D. Underwater Acoustic Digital Communication Channel for an UROV. In Proceedings of the 5th Annual Research Symposium (ARS2018), Hapugala, Sri Lanka, 4 January 2018; p. E17.
428. Wijesekara, P.A.D.S.N.; Sudheera, K.L.K.; Sandamali, G.G.N.; Chong, P.H.J. Data Gathering Optimization in Hybrid Software Defined Vehicular Networks. In Proceedings of the 20th Academic Sessions, Matara, Sri Lanka, 7 June 2023; p. 59.
429. Chiper, F.L.; Martian, A.; Vladeanu, C.; Marghescu, I.; Craciunescu, R.; Fratu, O. Drone detection and defense systems: Survey and a software-defined radio-based solution. *Sensors* **2022**, *22*, 1453. [CrossRef]
430. Zhang, K.; Zheng, G.; Wang, H.; Zhang, C.; Yu, X. Channel Model and Performance Analysis for MIMO Systems with Single Leaky Coaxial Cable in Tunnel Scenarios. *Sensors* **2022**, *22*, 5776. [CrossRef] [PubMed]
431. Grzechca, D.; Zieliński, D.; Filipowski, W. What is the effect of outer jacket degradation on the communication parameters? A case study of the twisted pair cable applied in the railway industry. *Energies* **2021**, *14*, 972. [CrossRef]
432. Aleksic, S. A survey on optical technologies for IoT, smart industry, and smart infrastructures. *J. Sens. Actuator Netw.* **2019**, *8*, 47. [CrossRef]
433. Burdin, V.A.; Dashkov, M.V.; Demidov, V.V.; Dukelskii, K.V.; Evtushenko, A.S.; Kuznetsov, A.A.; Matrosova, A.S.; Morozov, O.G.; Ter-Nersesyants, E.V.; Vasilets, A.A.; et al. New silica laser-optimized multimode optical fibers with extremely enlarged 100-μm core diameter for gigabit onboard and industrial networks. *Fibers* **2020**, *8*, 18. [CrossRef]
434. Morana, A.; Campanella, C.; Vidalot, J.; De Michele, V.; Marin, E.; Reghioua, I.; Boukenter, A.; Ouerdane, Y.; Paillet, P.; Girard, S. Extreme radiation sensitivity of ultra-low loss pure-silica-core optical fibers at low dose levels and infrared wavelengths. *Sensors* **2020**, *20*, 7254. [CrossRef]
435. Atzeni, D.; Bacciu, D.; Mazzei, D.; Prencipe, G. A Systematic Review of Wi-Fi and Machine Learning Integration with Topic Modeling Techniques. *Sensors* **2022**, *22*, 4925. [CrossRef]
436. Imoize, A.L.; Adedeji, O.; Tandiya, N.; Shetty, S. 6G enabled smart infrastructure for sustainable society: Opportunities, challenges, and research roadmap. *Sensors* **2021**, *21*, 1709. [CrossRef] [PubMed]
437. Alsharif, M.H.; Kelechi, A.H.; Albreem, M.A.; Chaudhry, S.A.; Zia, M.S.; Kim, S. Sixth generation (6G) wireless networks: Vision, research activities, challenges and potential solutions. *Symmetry* **2020**, *12*, 676. [CrossRef]
438. Li, T.; Chen, J.; Fu, H. Application scenarios based on SDN: An overview. *J. Phys. Conf. Ser.* **2019**, *1187*, 052067. [CrossRef]
439. Akyildiz, I.F.; Lee, A.; Wang, P.; Luo, M.; Chou, W. A roadmap for traffic engineering in SDN-OpenFlow networks. *Comput. Netw.* **2014**, *71*, 1–30. [CrossRef]
440. Yoo, Y.; Yang, G.; Kang, M.; Yoo, C. Adaptive control channel traffic shaping for virtualized SDN in clouds. In Proceedings of the 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), Beijing, China, 19–23 October 2020; pp. 22–24.
441. Veisi, F.; Montavont, J.; Theoleyre, F. SDN-TSCH: Enabling Software Defined Networking for Scheduled Wireless Networks with Traffic Isolation. In Proceedings of the 2022 IEEE Symposium on Computers and Communications (ISCC), Rhodes, Greece, 30 June–3 July 2022; pp. 1–7.
442. Pasca, S.T.V.; Kodali, S.S.P.; Kataoka, K. AMPS: Application aware multipath flow routing using machine learning in SDN. In Proceedings of the 2017 Twenty-third National Conference on Communications (NCC), Chennai, India, 2–4 March 2017; pp. 1–6.
443. Pham, T.A.Q.; Hadjadj-Aoul, Y.; Outtagarts, A. Deep reinforcement learning based qos-aware routing in knowledge-defined networking. In *Quality, Reliability, Security and Robustness in Heterogeneous Systems: 14th EAI International Conference, Ho Chi Minh City, Vietnam, 3–4 December 2018*; Proceedings 14; Springer International Publishing: Cham, Switzerland, 2019; pp. 14–26.

444. Cheng, L.C.; Wang, K.; Hsu, Y.H. Application-aware routing scheme for SDN-based cloud datacenters. In Proceedings of the 2015 Seventh International Conference on Ubiquitous and Future Networks, Sapporo, Japan, 7–10 July 2015; pp. 820–825.
445. Chahlaoui, F.; Dahmouni, H. A taxonomy of load balancing mechanisms in centralized and distributed SDN architectures. *SN Comput. Sci.* **2020**, *1*, 268. [CrossRef]
446. Zakia, U.; Yedder, H.B. Dynamic load balancing in SDN-based data center networks. In Proceedings of the 2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 3–5 October 2017; pp. 242–247.
447. Hai, N.T.; Kim, D.S. Efficient load balancing for multi-controller in SDN-based mission-critical networks. In Proceedings of the 2016 IEEE 14th International Conference on Industrial Informatics (INDIN), Poitiers, France, 19–21 July 2016; pp. 420–425.
448. Chen, J.; Wang, Y.; Huang, X.; Xie, X.; Zhang, H.; Lu, X. ALBLP: Adaptive load-balancing architecture based on link-state prediction in software-defined networking. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 8354150. [CrossRef]
449. Duy, P.T.; Qui, H.P.; Pham, V.H. Aloba: A mechanism of adaptive load balancing and failure recovery in distributed SDN controllers. In Proceedings of the 2019 IEEE 19th International Conference on Communication Technology (ICCT), Xi'an, China, 16–19 October 2019; pp. 1322–1326.
450. Kang, B.; Choo, H. An SDN-enhanced load-balancing technique in the cloud system. *J. Supercomput.* **2018**, *74*, 5706–5729. [CrossRef]
451. Abdelltif, A.A.; Ahmed, E.; Fong, A.T.; Gani, A.; Imran, M. SDN-based load balancing service for cloud servers. *IEEE Commun. Mag.* **2018**, *56*, 106–111. [CrossRef]
452. Yu, T.F.; Wang, K.; Hsu, Y.H. Adaptive routing for video streaming with QoS support over SDN networks. In Proceedings of the 2015 International Conference on Information Networking (ICOIN), Siem Reap, Cambodia, 12–14 January 2015; pp. 318–323.
453. Zheng, W.; Yang, M.; Zhang, C.; Zheng, Y.; Wu, Y.; Zhang, Y.; Li, J. Application-aware QoS routing in SDNs using machine learning techniques. *Peer-to-Peer Netw. Appl.* **2022**, *15*, 529–548. [CrossRef]
454. Nde, G.N.; Khondoker, R. SDN testing and debugging tools: A survey. In Proceedings of the 2016 5th International Conference on Informatics, Electronics and Vision (ICIEV), Dhaka, Bangladesh, 13–14 May 2016; pp. 631–635.
455. Zhang, Y.; Beheshti, N.; Manghirmalani, R. NetRevert: Rollback recovery in SDN. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 22 August 2014; pp. 231–232.
456. Cascone, C.; Pollini, L.; Sanvito, D.; Capone, A.; Sanso, B. SPIDER: Fault resilient SDN pipeline with recovery delay guarantees. In Proceedings of the 2016 IEEE NetSoft Conference and Workshops (NetSoft), Seoul, Republic of Korea, 6–10 June 2016; pp. 296–302.
457. Chandrasekaran, B.; Tschaen, B.; Benson, T. Isolating and tolerating SDN application failures with LegoSDN. In Proceedings of the Symposium on SDN Research, Santa Clara, CA, USA, 14–15 March 2016; pp. 1–12.
458. Li, Y.; Wang, Z.; Yao, J.; Yin, X.; Shi, X.; Wu, J.; Zhang, H. MSAID: Automated detection of interference in multiple SDN applications. *Comput. Netw.* **2019**, *153*, 49–62. [CrossRef]
459. Gheorghe, G.; Avanesov, T.; Palattella, M.R.; Engel, T.; Popoviciu, C. SDN-RADAR: Network troubleshooting combining user experience and SDN capabilities. In Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), London, UK, 13–17 April 2015; pp. 1–5.
460. Reitblatt, M.; Canini, M.; Guha, A.; Foster, N. Fattire: Declarative fault tolerance for software-defined networks. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, China, 16 August 2013; pp. 109–114.
461. Bi, Y.; Han, G.; Lin, C.; Guizani, M.; Wang, X. Mobility management for intro/inter domain handover in software-defined networks. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 1739–1754. [CrossRef]
462. Yao, D.; Su, X.; Liu, B.; Zeng, J. A mobile handover mechanism based on fuzzy logic and MPTCP protocol under SDN architecture. In Proceedings of the 2018 18th International Symposium on Communications and Information Technologies (ISCIT), Bangkok, Thailand, 26–28 September 2018; pp. 141–146.
463. Vieira, J.L.; Passos, D. An SDN-based access point virtualization solution for multichannel IEEE 802.11 networks. In Proceedings of the 2019 10th International Conference on Networks of the Future (NoF), Rome, Italy, 1–3 October 2019; pp. 122–125.
464. Lei, J.; Wang, Y.; Xia, Y. SDN-based centralized downlink scheduling with multiple aps cooperation in wlans. *Wirel. Commun. Mob. Comput.* **2019**, *2019*, 4015262. [CrossRef]
465. El Azaly, N.M.; Badran, E.F.; Kheirallah, H.N.; Farag, H.H. Centralized dynamic channel reservation mechanism via SDN for CR networks spectrum allocation. *IEEE Access* **2020**, *8*, 192493–192505. [CrossRef]
466. Yang, G.; Cao, Y.; Esmailpour, A.; Wang, D. SDN-Based Hierarchical Agglomerative Clustering Algorithm for Interference Mitigation in Ultra-Dense Small Cell Networks. *ETRI J.* **2018**, *40*, 227–236. [CrossRef]
467. Cheng, R.S.; Huang, C.M.; Pan, S.Y. WiFi offloading using the device-to-device (D2D) communication paradigm based on the software defined network (SDN) architecture. *J. Netw. Comput. Appl.* **2018**, *112*, 18–28. [CrossRef]
468. Rahimi, P.; Chrysostomou, C.; Pervaiz, H.; Vassiliou, V.; Ni, Q. Joint radio resource allocation and beamforming optimization for industrial internet of things in software-defined networking-based virtual fog-radio access network 5G-and-beyond wireless environments. *IEEE Trans. Ind. Inform.* **2021**, *18*, 4198–4209. [CrossRef]
469. Shantharama, P.; Thyagaturu, A.S.; Karakoc, N.; Ferrari, L.; Reisslein, M.; Scaglione, A. LayBack: SDN management of multi-access edge computing (MEC) for network access services and radio resource sharing. *IEEE Access* **2018**, *6*, 57545–57561. [CrossRef]

470. Bansal, M.; Mehlman, J.; Katti, S.; Levis, P. Openradio: A programmable wireless dataplane. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks, Helsinki, Finland, 13 August 2012; pp. 109–114.
471. Arnaz, A.; Lipman, J.; Abolhasan, M.; Hiltunen, M. Towards Integrating Intelligence and Programmability in Open Radio Access Networks: A Comprehensive Survey. *IEEE Access* **2022**, *10*, 67747–67770. [CrossRef]
472. Wang, Y.; Chen, H.; Wu, X.; Shu, L. An energy-efficient SDN based sleep scheduling algorithm for WSNs. *J. Netw. Comput. Appl.* **2016**, *59*, 39–45. [CrossRef]
473. Huang, H.; Guo, S.; Wu, J.; Li, J. Green datapath for TCAM-based software-defined networks. *IEEE Commun. Mag.* **2016**, *54*, 194–201. [CrossRef]
474. Huin, N.; Rifai, M.; Giroire, F.; Pacheco, D.L.; Urvoy-Keller, G.; Moulierac, J. Bringing energy aware routing closer to reality with SDN hybrid networks. *IEEE Trans. Green Commun. Netw.* **2018**, *2*, 1128–1139. [CrossRef]
475. De Assunção, M.D.; Carpa, R.; Lefévre, L.; Glück, O. On designing SDN services for energy-aware traffic engineering. In *Testbeds and Research Infrastructures for the Development of Networks and Communities: 11th International Conference, TRIDENTCOM, Hangzhou, China, 14–15 June 2016*; Revised Selected Papers; Springer International Publishing: Cham, Switzerland, 2017; pp. 14–23.
476. Pham, M.; Hoang, D.B.; Chaczko, Z. Congestion-aware and energy-aware virtual network embedding. *IEEE/ACM Trans. Netw.* **2019**, *28*, 210–223. [CrossRef]
477. Islam, M.J.; Rahman, A.; Kabir, S.; Karim, M.R.; Acharjee, U.K.; Nasir, M.K.; Band, S.S.; Sookhak, M.; Wu, S. Blockchain-SDN-based energy-aware and distributed secure architecture for IoT in smart cities. *IEEE Internet Things J.* **2021**, *9*, 3850–3864. [CrossRef]
478. Saha, D.; Shojaee, M.; Baddeley, M.; Haque, I. An energy-aware SDN/NFV architecture for the internet of things. In Proceedings of the 2020 IFIP Networking Conference (Networking), Paris, France, 22–25 June 2020; pp. 604–608.
479. Sellami, B.; Hakiri, A.; Yahia, S.B.; Berthou, P. Energy-aware task scheduling and offloading using deep reinforcement learning in SDN-enabled IoT network. *Comput. Netw.* **2022**, *210*, 108957. [CrossRef]
480. Klaedtke, F.; Karame, G.O.; Bifulco, R.; Cui, H. Access control for SDN controllers. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 22 August 2014; pp. 219–220.
481. Alsmadi, I. The integration of access control levels based on SDN. *Int. J. High Perform. Comput. Netw.* **2016**, *9*, 281–290. [CrossRef]
482. Tseng, Y.; Pattaranantakul, M.; He, R.; Zhang, Z.; Naït-Abdesselam, F. Controller DAC: Securing SDN controller with dynamic access control. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–6.
483. Hu, T.; Zhang, Z.; Yi, P.; Liang, D.; Li, Z.; Ren, Q.; Hu, Y.; Lan, J. SEAPP: A secure application management framework based on REST API access control in SDN-enabled cloud environment. *J. Parallel Distrib. Comput.* **2021**, *147*, 108–123. [CrossRef]
484. Chang, D.; Sun, W.; Yang, Y.; Wang, T. An E-ABAC-based SDN access control method. In Proceedings of the 2019 6th International Conference on Information Science and Control Engineering (ICISCE), Shanghai, China, 20–22 December 2019; pp. 668–672.
485. Weng, J.S.; Weng, J.; Zhang, Y.; Luo, W.; Lan, W. BENBI: Scalable and dynamic access control on the northbound interface of SDN-based VANET. *IEEE Trans. Veh. Technol.* **2018**, *68*, 822–831. [CrossRef]
486. Ren, W.; Sun, Y.; Luo, H.; Guizani, M. SILedger: A blockchain and ABE-based access control for applications in SDN-IoT networks. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 4406–4419. [CrossRef]
487. Bhattacharya, A.; Rana, R.; Datta, S.; Venkanna, U. P4-sKnock: A Two Level Host Authentication and Access Control Mechanism in P4 based SDN. In Proceedings of the 2022 27th Asia Pacific Conference on Communications (APCC), Jeju Island, Republisc of Korea, 19–21 October 2022; pp. 278–283.
488. Al-Alaj, A.; Sandhu, R.; Krishnan, R. A Model for the Administration of Access Control in Software Defined Networking using Custom Permissions. In Proceedings of the 2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), Atlanta, GA, USA, 28–31 October 2020; pp. 169–178.
489. Khalid, M.; Hameed, S.; Qadir, A.; Shah, S.A.; Draheim, D. Towards SDN-based smart contract solution for IoT access control. *Comput. Commun.* **2023**, *198*, 1–31. [CrossRef]
490. Toshniwal, B.; Joshi, K.D.; Shrivastava, P.; Kataoka, K. BEAM: Behavior-based access control mechanism for SDN applications. In Proceedings of the 2019 28th International Conference on Computer Communication and Networks (ICCCN), Valencia, Spain, 29 July–1 August 2019; pp. 1–2.
491. Chattaraj, D.; Bera, B.; Das, A.K.; Rodrigues, J.J.; Park, Y. Designing fine-grained access control for software-defined networks using private blockchain. *IEEE Internet Things J.* **2021**, *9*, 1542–1559. [CrossRef]
492. Bensalah, F.; El Kamoun, N. A novel approach for improving MPLS VPN security by adopting the software defined network paradigm. *Procedia Comput. Sci.* **2019**, *160*, 831–836. [CrossRef]
493. Shif, L.; Wang, F.; Lung, C.H. Improvement of security and scalability for IoT network using SD-VPN. In Proceedings of the NOMS 2018—2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 23–27 April 2018; pp. 1–5.
494. Hauser, F.; Häberle, M.; Schmidt, M.; Menth, M. P4-ipsec: Site-to-site and host-to-site vpn with ipsec in p4-based sdn. *IEEE Access* **2020**, *8*, 139567–139586. [CrossRef]
495. Morzhov, S.; Alekseev, I.; Nikitinskiy, M. Firewall application for Floodlight SDN controller. In Proceedings of the 2016 International Siberian Conference on Control and Communications (SIBCON), Moscow, Russia, 12–14 May 2016; pp. 1–5.
496. Bakker, J.N.; Welch, I.; Seah, W.K. Network-wide virtual firewall using SDN/OpenFlow. In Proceedings of the 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, 7–10 November 2016; pp. 62–68.

497. Monir, M.F.; Pan, D. Application and Assessment of Click Modular Firewall vs. POX Firewall in SDN/NFV Framework. In Proceedings of the 2020 IEEE Region 10 Conference (Tencon), Osaka, Japan, 16–19 November 2020; pp. 991–996.
498. Prabakaran, S.; Ramar, R.; Hussain, I.; Kavin, B.P.; Alshamrani, S.S.; AlGhamdi, A.S.; Alshehri, A. Predicting attack pattern via machine learning by exploiting stateful firewall as virtual network function in an SDN network. *Sensors* **2022**, *22*, 709. [CrossRef]
499. Tran, T.V.; Ahn, H. FlowTracker: A SDN stateful firewall solution with adaptive connection tracking and minimized controller processing. In Proceedings of the 2016 International Conference on Software Networking (ICSN), Jeju Island, Republic of Korea, 23–26 May 2016; pp. 1–5.
500. Kavin, B.P.; Srividhya, S.R.; V, R.; Lai, W.C. Performance Evaluation of Stateful Firewall-Enabled SDN with Flow-Based Scheduling for Distributed Controllers. *Electronics* **2022**, *11*, 3000.
501. Shirali-Shahreza, S.; Ganjali, Y. Protecting home user devices with an SDN-based firewall. *IEEE Trans. Consum. Electron.* **2018**, *64*, 92–100. [CrossRef]
502. Steichen, M.; Hommes, S.; State, R. ChainGuard—A firewall for blockchain applications using SDN with OpenFlow. In Proceedings of the 2017 Principles, Systems and Applications of IP Telecommunications (IPTComm), Chicago, IL, USA, 25–28 September 2017; pp. 1–8.
503. Chang, Y.W.; Lin, T.N. An efficient dynamic rule placement for distributed firewall in sdn. In Proceedings of the GLOBECOM 2020—2020 IEEE Global Communications Conference, Taipei, Taiwan, 7–11 December 2020; pp. 1–6.
504. Datta, R.; Choi, S.; Chowdhary, A.; Park, Y. P4guard: Designing p4 based firewall. In Proceedings of the MILCOM 2018—2018 IEEE Military Communications Conference (MILCOM), Los Angeles, CA, USA, 29–31 October 2018; pp. 1–6.
505. Wijesekara, P.A.D.S.N. Prevalence, Risk Factors and Remedies for Psychiatric Illnesses among Students in Higher Education: A Comprehensive Study in University of Ruhuna. *Prepr. Res. Sq.* **2022**, 1–33.
506. Birkinshaw, C.; Rouka, E.; Vassilakis, V.G. Implementing an intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks. *J. Netw. Comput. Appl.* **2019**, *136*, 71–85. [CrossRef]
507. Latah, M.; Toker, L. Towards an efficient anomaly-based intrusion detection for software-defined networks. *IET Netw.* **2018**, *7*, 453–459. [CrossRef]
508. Zeleke, E.M.; Melaku, H.M.; Mengistu, F.G. Efficient intrusion detection system for SDN orchestrated Internet of Things. *J. Comput. Netw. Commun.* **2021**, *2021*, 5593214. [CrossRef]
509. Segura, G.A.N.; Chorti, A.; Margi, C.B. Centralized and distributed intrusion detection for resource-constrained wireless SDN networks. *IEEE Internet Things J.* **2021**, *9*, 7746–7758. [CrossRef]
510. Li, W.; Wang, Y.; Jin, Z.; Yu, K.; Li, J.; Xiang, Y. Challenge-based collaborative intrusion detection in software-defined networking: An evaluation. *Dig. Commun. Netw.* **2021**, *7*, 257–263. [CrossRef]
511. Shu, J.; Zhou, L.; Zhang, W.; Du, X.; Guizani, M. Collaborative intrusion detection for VANETs: A deep learning-based distributed SDN approach. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 4519–4530. [CrossRef]
512. Ujjan, R.M.A.; Pervez, Z.; Dahal, K. Snort based collaborative intrusion detection system using blockchain in SDN. In Proceedings of the 2019 13th International Conference on Software, Knowledge, Information Management and Applications (SKIMA), Island of Ulkulhas, Maldives, 26–28 August 2019; pp. 1–8.
513. Hurley, T.; Perdomo, J.E.; Perez-Pons, A. HMM-based intrusion detection system for software defined networking. In Proceedings of the 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), Anaheim, CA, USA, 18–20 December 2016; pp. 617–621.
514. Tang, T.A.; McLernon, D.; Mhamdi, L.; Zaidi, S.A.R.; Ghogho, M. Intrusion detection in sdn-based networks: Deep recurrent neural network approach. In *Deep Learning Applications for Cyber Security*; Springer: Cham, Switzerland, 2019; pp. 175–195.
515. Bhatia, A.; Haribabu, K.; Gupta, K.; Sahu, A. Realization of flexible and scalable VANETs through SDN and virtualization. In Proceedings of the 2018 International conference on information networking (ICOIN), Bangkok, Thailand, 10–12 January 2018; pp. 280–282.
516. Kurniawan, M.T.; Fathinuddin, M.; Widiyanti, H.A.; Simanjuntak, G.R. Network Slicing on SDN using FlowVisor and POX Controller to Traffic Isolation Enforcement. In Proceedings of the 2021 International Conference on Engineering and Emerging Technologies (ICEET), Istanbul, Turkey, 27–28 October 2021; IEEE: Toulouse, France, 2021; pp. 1–6.
517. Chen, C.H.; Chen, C.; Lu, S.H.; Tseng, C.C. Role-based campus network slicing. In Proceedings of the 2016 IEEE 24th International Conference on Network Protocols (ICNP), Singapore, 8–11 November 201; pp. 1–6.
518. Zhang, N.; Yang, P.; Zhang, S.; Chen, D.; Zhuang, W.; Liang, B.; Shen, X.S. Software defined networking enabled wireless network virtualization: Challenges and solutions. *IEEE Netw.* **2017**, *31*, 42–49. [CrossRef]
519. Ma, Y.W.; Chen, J.L.; Chang, C.C.; Nakao, A.; Yamamoto, S. A novel dynamic resource adjustment architecture for virtual tenant networks in SDN. *J. Syst. Softw.* **2018**, *143*, 100–115. [CrossRef]
520. Wang, Q.; Shou, G.; Liu, Y.; Hu, Y.; Guo, Z.; Chang, W. Implementation of multipath network virtualization with SDN and NFV. *IEEE Access* **2018**, *6*, 32460–32470. [CrossRef]
521. Han, Y.; Vachuska, T.; Al-Shabibi, A.; Li, J.; Huang, H.; Snow, W.; Hong, J.W.K. ONVisor: Towards a scalable and flexible SDN-based network virtualization platform on ONOS. *Int. J. Netw. Manag.* **2018**, *28*, e2012. [CrossRef]
522. Shi, J.; Chung, S.H. A traffic-aware quality-of-service control mechanism for software-defined networking-based virtualized networks. *Int. J. Distrib. Sens. Netw.* **2017**, *13*, 1550147717697984. [CrossRef]

523. Rawat, D.B. Fusion of software defined networking, edge computing, and blockchain technology for wireless network virtualization. *IEEE Commun. Mag.* **2019**, *57*, 50–55. [CrossRef]
524. Cui, L.; Yu, F.R.; Yan, Q. When big data meets software-defined networking: SDN for big data and big data for SDN. *IEEE Netw.* **2016**, *30*, 58–65. [CrossRef]
525. Aujla, G.S.; Kumar, N.; Zomaya, A.Y.; Ranjan, R. Optimal decision making for big data processing at edge-cloud environment: An SDN perspective. *IEEE Trans. Ind. Inform.* **2017**, *14*, 778–789. [CrossRef]
526. Cho, J.; Chang, H.; Mukherjee, S.; Lakshman, T.V.; Van der Merwe, J. Typhoon: An SDN enhanced real-time big data streaming framework. In Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies, Incheon, Republic of Korea, 12–15 November 2017; pp. 310–322.
527. Alwasel, K.; Calheiros, R.N.; Garg, S.; Buyya, R.; Pathan, M.; Georgakopoulos, D.; Ranjan, R. BigDataSDNSim: A simulator for analyzing big data applications in software-defined cloud data centers. *Softw. Pract. Exp.* **2021**, *51*, 893–920. [CrossRef]
528. Shah, S.A.R.; Wu, W.; Lu, Q.; Zhang, L.; Sasidharan, S.; DeMar, P.; Guok, C.; Macauley, J.; Pouyoul, E.; Kim, J.; et al. AmoebaNet: An SDN-enabled network service for big data science. *J. Netw. Comput. Appl.* **2018**, *119*, 70–82. [CrossRef]
529. Jain, S.; Khandelwal, M.; Katkar, A.; Nygate, J. Applying big data technologies to manage QoS in an SDN. In Proceedings of the 2016 12th International Conference on Network and Service Management (CNSM), Montreal, QC, Canada, 31 October–4 November 2016; pp. 302–306.
530. Trevisan, M.; Drago, I.; Mellia, M.; Song, H.H.; Baldi, M. AWESoME: Big data for automatic Web service management in SDN. *IEEE Trans. Netw. Serv. Manag.* **2017**, *15*, 13–26. [CrossRef]
531. Le, L.V.; Sinh, D.; Lin, B.S.P.; Tung, L.P. Applying big data, machine learning, and SDN/NFV to 5G traffic clustering, forecasting, and management. In Proceedings of the 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), Montreal, QC, Canada, 25–29 June 2018; pp. 168–176.
532. Le, L.V.; Lin, B.S.P.; Tung, L.P.; Sinh, D. SDN/NFV, machine learning, and big data driven network slicing for 5G. In Proceedings of the 2018 IEEE 5G World Forum (5GWF), Silicon Valley, CA, USA, 9–11 July 2018; pp. 20–25.
533. Lv, Z.; Xiu, W. Interaction of edge-cloud computing based on SDN and NFV for next generation IoT. *IEEE Internet Things J.* **2019**, *7*, 5706–5712. [CrossRef]
534. Mayoral, A.; Vilalta, R.; Muñoz, R.; Casellas, R.; Martínez, R. SDN orchestration architectures and their integration with cloud computing applications. *Opt. Switch. Netw.* **2017**, *26*, 2–13. [CrossRef]
535. Rawas, S. Energy, network, and application-aware virtual machine placement model in SDN-enabled large scale cloud data centers. *Multimed. Tools Appl.* **2021**, *80*, 15541–15562. [CrossRef]
536. Li, F.; Cao, J.; Wang, X.; Sun, Y.; Sahni, Y. Enabling software defined networking with qos guarantee for cloud applications. In Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, HI, USA, 25–30 June 2017; pp. 130–137.
537. Conti, M.; Kaliyar, P.; Lal, C. CENSOR: Cloud-enabled secure IoT architecture over SDN paradigm. *Concurr. Comput. Pract. Exp.* **2019**, *31*, e4978. [CrossRef]
538. Rahman, A.; Islam, M.J.; Khan, M.S.I.; Kabir, S.; Pritom, A.I.; Karim, M.R. Block-sdotcloud: Enhancing security of cloud storage through blockchain-based sdn in iot network. In Proceedings of the 2020 2nd International Conference on Sustainable Technologies for Industry 4.0 (STI), Dhaka, Bangladesh, 19–20 December 2020; pp. 1–6.
539. Gargees, R.; Morago, B.; Pelapur, R.; Chemodanov, D.; Calyam, P.; Oraibi, Z.; Duan, Y.; Seetharaman, G.; Palaniappan, K. Incident-supporting visual cloud computing utilizing software-defined networking. *IEEE Trans. Circuits Syst. Video Technol.* **2016**, *27*, 182–197. [CrossRef]
540. Liu, Y.; Gu, H.; Yan, F.; Calabretta, N. Highly-efficient switch migration for controller load balancing in elastic optical inter-datacenter networks. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 2748–2761. [CrossRef]
541. Song, P.; Liu, Y.; Liu, T.; Qian, D. Controller-proxy: Scaling network management for large-scale SDN networks. *Comput. Commun.* **2017**, *108*, 52–63. [CrossRef]
542. Lu, Y.; Ling, Z.; Zhu, S.; Tang, L. SDTCP: Towards datacenter TCP congestion control with SDN for IoT applications. *Sensors* **2017**, *17*, 109. [CrossRef] [PubMed]
543. Paliwal, M.; Shrimankar, D. Effective resource management in SDN enabled data center network based on traffic demand. *IEEE Access* **2019**, *7*, 69698–69706. [CrossRef]
544. Pham, M.; Hoang, D.B. SDN applications—The intent-based Northbound Interface realisation for extended applications. In Proceedings of the 2016 IEEE NetSoft Conference and Workshops (NetSoft), Seoul, Republic of Kprea, 6–10 June 2016; pp. 372–377.
545. Siniarski, B.; Murphy, J.; Delaney, D. FlowVista: Low-bandwidth SDN monitoring driven by business application interaction. In Proceedings of the 2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 21–23 September 2017; pp. 1–6.
546. Krishna, S.V.; Shrivastava, A.; Wagh, S.J. SDN in high performance computing for scien. In Proceedings of the 2017 International Conference on Computational Intelligence in Data Science (ICCIDS), Chennai, India, 2–3 June 2017; pp. 1–8.
547. Rauf, B.; Abbas, H.; Sheri, A.M.; Iqbal, W.; Khan, A.W. Enterprise integration patterns in SDN: A reliable, fault-tolerant communication framework. *IEEE Internet Things J.* **2020**, *8*, 6359–6371. [CrossRef]

548. Rauf, B.; Abbas, H.; Sheri, A.M.; Iqbal, W.; Bangash, Y.A.; Daneshmand, M.; Amjad, M.F. CACS: A context-aware and anonymous communication framework for an enterprise network using SDN. *IEEE Internet Things J.* **2021**, *9*, 11725–11736. [CrossRef]
549. Van Adrichem, N.L.; Iqbal, F.; Kuipers, F.A. Backup rules in software-defined networks. In Proceedings of the 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, 7–10 November 2016; pp. 179–185.
550. Zhang, P.; Li, H.; Hu, C.; Hu, L.; Xiong, L. Stick to the script: Monitoring the policy compliance of SDN data plane. In Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems, Santa Clara, CA, USA, 17–18 March 2016; pp. 81–86.
551. Dangovas, V.; Kuliesius, F. SDN-driven authentication and access control system. In Proceedings of the International Conference on Digital Information, Networking, and Wireless Communications (DINWC2014), Ostrava, Czech Republic, 24–26 June 2014; pp. 20–23.
552. Wen, X.; Yang, B.; Chen, Y.; Li, L.E.; Bu, K.; Zheng, P.; Yang, Y.; Hu, C. RuleTris: Minimizing rule update latency for TCAM-based SDN switches. In Proceedings of the 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), Nara, Japan, 27–30 June 2016; pp. 179–188.
553. Wani, M.A.; Bhat, F.A.; Afzal, S.; Khan, A.I. *Advances in Deep Learning*; Springer: Singapore, 2020.
554. Khan, P.W.; Byun, Y.C.; Park, N. IoT-blockchain enabled optimized provenance system for food industry 4.0 using advanced deep learning. *Sensors* **2020**, *20*, 2990. [CrossRef]
555. Luo, Y.; Cascon, P.; Murray, E.; Ortega, J. Accelerating OpenFlow switching with network processors. In Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, Princeton, NJ, USA, 19–20 October 2009; pp. 70–71.
556. Hang, Z.; Wen, M.; Shi, Y.; Zhang, C. Programming protocol-independent packet processors high-level programming (P4HLP): Towards unified high-level programming for a commodity programmable switch. *Electronics* **2019**, *8*, 958. [CrossRef]
557. Isyaku, B.; Mohd Zahid, M.S.; Bte Kamat, M.; Abu Bakar, K.; Ghaleb, F.A. Software defined networking flow table management of openflow switches performance and security challenges: A survey. *Future Internet* **2020**, *12*, 147. [CrossRef]
558. Chica, J.C.C.; Imbachi, J.C.; Vega, J.F.B. Security in SDN: A comprehensive survey. *J. Netw. Comput. Appl.* **2020**, *159*, 102595. [CrossRef]
559. Diogo, M.; Cabral, B.; Bernardino, J. Consistency models of NoSQL databases. *Future Internet* **2019**, *11*, 43. [CrossRef]
560. Filip, I.D.; Iliescu, C.M.; Pop, F. Assertive, Selective, Scalable IoT-Based Warning System. *Sensors* **2022**, *22*, 1015. [CrossRef] [PubMed]
561. Bakarić, R.; Korenčić, D.; Hršak, D.; Ristov, S. SFQ: Constructing and Querying a Succinct Representation of FASTQ Files. *Electronics* **2022**, *11*, 1783. [CrossRef]
562. Saeed, A.S.M.; George, L.E. Data deduplication system based on content-defined chunking using bytes pair frequency occurrence. *Symmetry* **2020**, *12*, 1841. [CrossRef]
563. Voellmy, A.; Wang, J. Scalable software defined network controllers. In Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Helsinki, Finland, 13–17 August 2012; pp. 289–290.
564. Blenk, A.; Basta, A.; Reisslein, M.; Kellerer, W. Survey on network virtualization hypervisors for software defined networking. *IEEE Commun. Surv. Tutor.* **2015**, *18*, 655–685. [CrossRef]
565. Horvath, T.; Munster, P.; Oujezsky, V.; Bao, N.H. Passive optical networks progress: A tutorial. *Electronics* **2020**, *9*, 1081. [CrossRef]
566. Hoebeke, J.; Moerman, I.; Dhoedt, B.; Demeester, P. An overview of mobile ad hoc networks: Applications and challenges. *J.-Commun. Netw.* **2004**, *3*, 60–66.
567. Chlamtac, I.; Conti, M.; Liu, J.J.N. Mobile ad hoc networking: Imperatives and challenges. *Ad Hoc Netw.* **2003**, *1*, 13–64. [CrossRef]
568. Majid, M.; Habib, S.; Javed, A.R.; Rizwan, M.; Srivastava, G.; Gadekallu, T.R.; Lin, J.C.W. Applications of wireless sensor networks and internet of things frameworks in the industry revolution 4.0: A systematic literature review. *Sensors* **2022**, *22*, 2087. [CrossRef]
569. Gupta, C.; Johri, I.; Srinivasan, K.; Hu, Y.C.; Qaisar, S.M.; Huang, K.Y. A systematic review on machine learning and deep learning models for electronic information security in mobile networks. *Sensors* **2022**, *22*, 2017. [CrossRef]
570. *IEEE P2302-2021*; Standard for Intercloud Interoperability and Federation (SIIF): Reference Architecture and Taxonomy Framework. IEEE Standards Association: Piscataway, NJ, USA, 2021. Available online: https://standards.ieee.org/ieee/2302/7056/ (accessed on 14 March 2023).
571. *IEEE P2413-2019*; Standard for an Architectural Framework for the Internet of Things (IoT). IEEE Standards Association: Piscataway, NJ, USA, 2019. Available online: https://standards.ieee.org/ieee/2413/6226/ (accessed on 14 March 2023).
572. Novo, O.; Francesco, M.D. Semantic interoperability in the IoT: Extending the web of things architecture. *ACM Trans. Internet Things* **2020**, *1*, 1–25. [CrossRef]
573. Open Network Foundation. Stratum. Available online: https://opennetworking.org/stratum/ (accessed on 14 March 2023).

574. Metro Ethernet Forum. LSO SONATA. Available online: https://www.mef.net/service-automation/lso-apis/inter-provider-apis/lso-sonata/ (accessed on 14 March 2023).
575. International Telecommunication Union. Study Group 13 at a Glance. Available online: https://www.itu.int/en/ITU-T/about/groups/Pages/sg13.aspx (accessed on 14 March 2023).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.