applied sciences

Article

Artificial Intelligence Control Logic in Next-Generation Programmable Networks

Mateusz Żotkiewicz [1], Wiktor Szałyga [1], Jaroslaw Domaszewicz [1], Andrzej Bąk [1], Zbigniew Kopertowski [2] and Stanisław Kozdrowski [3,*]

[1] Institute of Telecommunications, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland; mzotkiew@tele.pw.edu.pl (M.Ż.); wiktor.szalyga.stud@pw.edu.pl (W.S.); domaszew@tele.pw.edu.pl (J.D.); bak@tele.pw.edu.pl (A.B.)
[2] Orange Labs Polska, Obrzezna 7, 02-691 Warszawa, Poland; Zbigniew.Kopertowski@orange.com
[3] Institute of Computer Science, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland
* Correspondence: s.kozdrowski@elka.pw.edu.pl

Abstract: The new generation of programmable networks allow mechanisms to be deployed for the efficient control of dynamic bandwidth allocation and ensure Quality of Service (QoS) in terms of Key Performance Indicators (KPIs) for delay or loss sensitive Internet of Things (IoT) services. To achieve flexible, dynamic and automated network resource management in Software-Defined Networking (SDN), Artificial Intelligence (AI) algorithms can provide an effective solution. In the paper, we propose the solution for network resources allocation, where the AI algorithm is responsible for controlling intent-based routing in SDN. The paper focuses on the problem of optimal switching of intents between two designated paths using the Deep-Q-Learning approach based on an artificial neural network. The proposed algorithm is the main novelty of this paper. The Developed Networked Application Emulation System (NAPES) allows the AI solution to be tested with different patterns to evaluate the performance of the proposed solution. The AI algorithm was trained to maximize the total throughput in the network and effective network utilization. The results presented confirm the validity of applied AI approach to the problem of improving network performance in next-generation networks and the usefulness of the NAPES traffic generator for efficient economical and technical deployment in IoT networking systems evaluation.

Keywords: artificial intelligence; deep-Q-learning; internet of things; software defined networking; programmable networks; IoT traffic generation

check for updates

Citation: Żotkiewicz, M.; Szałyga, W.; Domaszewicz, J.; Bąk, A.; Kopertowski, Z.; Kozdrowski, S. Artificial Intelligence Control Logic in Next Generation Programmable Networks. *Appl. Sci.* **2021**, *11*, 9163. https://doi.org/10.3390/app11199163

Academic Editor: Eui-Nam Huh

Received: 9 August 2021
Accepted: 23 September 2021
Published: 2 October 2021

1. Introduction

This paper presents a solution developed in the FlexNet (www.celticnext.eu/project-flexnet) project related to the management of network resources using Software-Defined Networking (SND) technology. SDN is a new networking solution in which a central server, called a controller, oversees all processes and controls network behaviour, ensuring the best possible network quality. This new paradigm of network design, as opposed to the traditional method, has benefits. It is much easier to adapt to new network policies using software, as it is easier to add or modify controller-based network-level rules using software, rather than manually applying a limited set of commands to these devices. The SDN model allows control functions found in network devices to be taken and moved centrally at the SDN controller level, allowing network devices to communicate with each other in an efficient manner.

Presented approach is in accordance to FlexNet project objective of building up a new paradigm of flexible network communications to foster IoT value creation. The Flexible IoT Network provides the IoT value creators with the availability to consume network communications on demand, in real time, and automatically to fulfil their specific needs.

This new network paradigm is fully aligned with the efforts currently ongoing in the implementation of 5G technology, providing high-quality and consistent connectivity for people and objects, creating the perception of infinite capacity. The FlexNet project proposes flexible resource management using SDN with the support of an AI solution for different IoT use cases.

Usually, the network is managed manually using commands, scripts or special tools, without automation for the efficient allocation of network resources. For several years, new network solutions with improved management solutions can be seen, where the most advanced one is the SDN solution [1–3]. In [4], the basic mechanisms and techniques of SDN for dynamic and scalable network control envisioned for 5G technology are defined [5]. One of the main elements is the SDN controller, which allows adaptive dynamic provisioning of resources by applying management rules to the traffic flow in the network [6]. On the other hand, the traffic generated in the network is also becoming more complex, especially in IoT applications [7–11] where the management of large data volumes requires more flexibility and scalability [12,13].

This flexibility and efficiency approach is often supported by artificial intelligence (AI) algorithms. In our proposed approach, the AI algorithm supports the intent routing control in SDN and is responsible for the resource allocation mechanism and network parameters such as throughput and network losses. This approach is also used in our ongoing FlexNet project [14–16]. The FlexNet project covers different use cases related to IoT technology, where different Quality of Service (QoS) requirements have to be met.

In addition, an IoT traffic generator called Networked Application Emulation System (NAPES) was developed and used for solution validation purposes. New applications and end-user devices generate different network traffic patterns; therefore, for validating new network solutions in complex application scenarios, the aforementioned IoT traffic generator is a very useful, cost-effective and fast-to-implement tool.

1.1. Motivation

The evolution of the next-generation telecommunication network in programmable flexible resources control solutions is driven by growing application requirements in service quality and networking resources [17]. Especially, it is the challenge in the IoT domain where multiply vertical use cases with different types of requirements are needed to be effectively implement. In the FlexNet project, we focused on different types of IoT use cases like:

- Emergency and public safety with low latency requests;
- Video surveillance applications, where on demand high bandwidth is required.

In these situations, SDN approach allows for flexible, on demand creation of new services using network controller handling virtualized resource across the network. Automatized management of the network resources in relation, from one side to applications demands and from other side to their optimal effective utilization is complex task, where AI mechanisms are promising solutions.

AI has seen a surge of interest in the networking community [18]. Recent contributions include data-driven flow control for wide-area networks, job scheduling, and network congestion control [19]. A particularly promising domain is the network management. Researchers have used Machine Learning (ML) to automate the management of network resources in relation to routing or network traffic optimization [20–24]. In loT networks, we expect network conditions to vary over time and space. Time varying conditions may be long term (seasonal) or short term, resulting in a significant impact on network performance [25]. ML techniques will be developed in order to detect such changes and signal them to the SDN layer for timely action to be taken to improve the overall network performance. Another solution is a Knowledge-Defined Network (KDN) which also is a next step on the path towards an implementation of a self-driving network [26]. KDN is a complementary solution for SDN that brings reasoning processes and ML techniques into the network control plane to enable autonomous and fast operation and minimization of operational costs.

1.2. Article Organisation

The article is organized as follows: in Section 2, we briefly describe the problem and application of AI in the contribution. Section 3 presents description of IoT traffic generator called NAPES we developed and use in the contribution. In Section 4, the results are presented. Contributions of this work are summarized and future work directions are discussed in Section 5.

2. Problem Description

The work has been done in the scope of FlexNet project, where Open Network Operating System (ONOS) controller has been used to control SDN. In ONOS, the concept of intents is used. An intent express a willingness to send a specific amount of data through a network [27].

In the considered architecture, for each registered intent, a pair of paths is computed. The paths are computed immediately after an intent is registered, and the process can take up to a couple of seconds. Then, when a network is in operational state, the AI module described in this paper selects one path from these precomputed paths for each intent. The selection itself has to be fast and prone to the unexpected behaviour of network links and intents. In other words, we face the problem of optimal switching of intents between two previously computed paths. The switching is performed based on an output of an artificial neural network that has been trained to recognize situations and states of the network requiring switching from one path to another.

Consider the following sets:

$\mathcal{E}$ edges

$\mathcal{I}$ intents

$\mathcal{P}_i$ paths for intent $i \in \mathcal{I}$

$\mathcal{E}_p$ edges used on path $p \in \mathcal{P}$

Each intent has its volume defined for each moment in time in the considered time horizon $< 0; T >$. However, the volume is not known in advance. Intents have to be assigned to available paths in a way that minimizes the congestion resulting from the limited capacity of the edges. The volume and the capacities are expressed using the following constants:

v_{it} volume of intent $i \in \mathcal{I}$ at time $t \in < 0; T >$

c_e capacity on edge $e \in \mathcal{E}$

The assignment and the congestion are expressed using the following variables:

$x_{it} \in \mathcal{P}_i$ equals the path intent $i \in \mathcal{I}$ uses at time $t \in < 0; T >$

y_{it} volume intent $i \in \mathcal{I}$ is actually sending at time $t \in < 0; T >$ due to congestion

We express congestion using the following function that depends on the network, loads, and selected routes:

$$y_{it} = \begin{cases} v_{it} & \forall e \in \mathcal{E}_{x_{it}} \sum_{i' : e \in \mathcal{E}_{x_{i't}}} v_{i't} \le c_e \\ C(\mathbf{x_t}, i) & \text{otherwise} \end{cases} \qquad \forall i \in \mathcal{I},\ \forall t \in < 0; T >$$

where $C(\mathbf{x_t}, i)$ is a non-trivial function that depends on the network, its current state, and an utilized congestion avoidance algorithm. The actual optimization problem we are solving is as follows:

$$\max \int_0^T \sum_{i \in \mathcal{I}} y_{it}$$

We notice that even a formally defined problem with unrealistically constant and predictable behaviour of intents and state of links is *NP*-hard, because we can reduce the satisfiability problem (SAT) [28] to it. Therefore, we decided first to decompose the problem to alleviate the computational complexity and second to use artificial neural networks to cope with unexpected behaviour of intents and links. The decomposition consists of considering each intent independently. The considered artificial neural network knows neither an exact topology of SDN nor volumes and paths of all intents. Instead, it is fed with the view of SDN from a point of view of a single intent that consists of the detailed information about the considered intent and the aggregated information about other intents in the vicinity.

We developed a system whose conceptual model is presented in Figure 1. First, it collects information about intents registered in ONOS using Intent Monitor and Reroute (IMR) service and standard ONOS API. Then, it collects information about traffic in a network using ifstat, which is an open-source, interface statistics collecting tool that is available in almost all Linux machines.

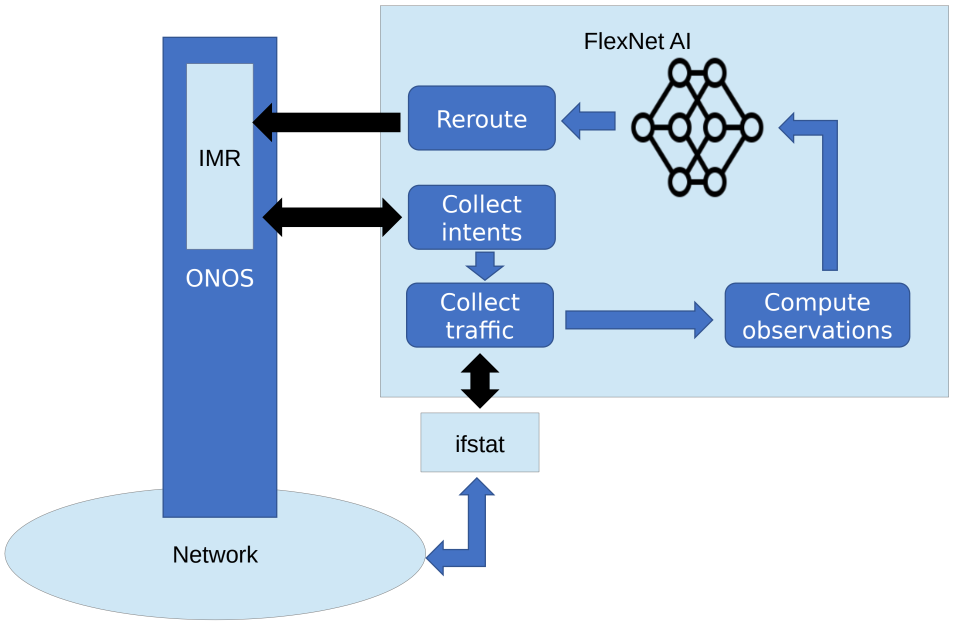

Figure 1. Conceptual model of the system.

In the next step, it combines this knowledge and creates observations for a randomly selected group of registered intent independently. The computed observations for each selected intent independently enter a neural network that returns two values associated with expected values of q-functions for two possible actions: doing nothing or switching to another path. If the latter value is greater than the former value, the switching is performed. In this way, in each step, only a part of registered intents is considered, and usually only a small fraction of this part of intents is issued with a command to change a current path.

The artificial neural network used in this research is a Deep-Q-Network (DQN) [29] consisting of four inputs (observations), two outputs, and four hidden layers of ten, five, five, and three neurons. We used DQNs, because it gave the best results in our internal studies. The learning process was implemented using Ray framework [30] and its goal was to maximize the total throughput in the network. The neural network takes a state of a network from a viewpoint of a single intent as an input and as an output it produces the expected q-function value for two cases: (a) when nothing happens and (b) when the path is switched.

The observations provided to the artificial neural network depend on a considered intent and are computed using data collected from a network and from ONOS. The observations are as follows:

- Ratio of efficiency on active path;
- Ratio of potential efficiency on inactive path;
- Edge occupancy percentage on active path;
- Edge occupancy percentage on inactive path.

The first observation is a ratio of an obtained momentum throughput for the intent to the maximum throughput declared for this intent. The maximum declared throughput is provided by the owner of the intent when creating the intent using the ONOS interface. The second observation is a ratio of the minimum available capacity on all edges of the inactive path to the declared intent's maximum throughput. The third observation is the minimum ratio of an obtained momentum throughput for the intent to a total traffic on an edge for all edges of the active path. Similarly, the fourth observation is the minimum ratio of the obtained momentum throughput for the intent to a total potential traffic (current traffic increases by the current momentum throughput of the intent) on an edge for all edges of the inactive path. All these observations are computed independently for each intent using data collected from ONOS and ifstat.

The artificial neural network that is fed with the above observations was trained using a heavily modified Iroko framework [31] that is based on Open AI Gym [32] and was originally implemented to optimize traffic in data centers. Episodes were run with the mininet framework using a modified version of goben [33] to generate various traffic patterns. The neural network was trained to maximize the total throughput in a network. The modifications to Iroko framework are as follows:

- Procedures to collect traffic from a network using ifstat;
- Procedures to compute the previously described observations from the traffic;
- Objective function fed to Ray that now expresses the total throughput in a network;
- Dynamically modified network topologies that can change during training between episodes;
- Dynamically selected currently considered intents that also change between episodes;
- Support for dynamic traffic that can vary between iterations.

The network was trained in series of episodes. Each episode was defined by an SDN network topology and a set of active traffic demands. Each demand was described by its source, destination, and an active time period. Note that the proposed neural network was trained using various topologies and traffic patters; thus, it is topology independent. Because of the limitations of mininet and utilized machines, the considered topologies consisted of ten switches at most. In the training process, we used three different topologies with eight different traffic patterns for each topology. The episodes were repeated for 36 times, totalling 864 episodes in the training process. Each episode lasted for 100 s in real time, resulting in 100 training steps. In each step, an active demand was selected at random and only this demand was considered in this particular step. The observations were computed for the selected demand, and depending on the output of a current neural network the action was taken.

We set the learning rate of RLLib's Adam optimizer [34] to 5×10^{-4}. A replay buffer size was set to 5000, which is approximately twice as much as a number of steps in one loop over all utilized topologies and traffic patterns. The exploration ϵ decreased linearly from 1.0 to 0.02 in the first 5000 steps and then remained constant. Finally, the target network was updated each 200 steps, and the training batch size was set to 8.

3. Generating Network Traffic

To test our AI solution, we use Networked Application Emulation System (NAPES), a traffic generator we developed within the FlexNet project. The motivation behind the development of NAPES is to enable rapid deployment of setups that exhibit complex, time varying traffic patterns, which closely resembles those of actual applications. The main innovative idea behind NAPES is that it allows distributed, traffic-generating "applications" to be defined with elements of application logic. Specifically, a NAPES application consists of communicating application components, each of which can have several state machines representing the component's logic. State machines are meant to approximate the logic of an actual application. Interacting state machines of an application's components jointly give rise to complex traffic patterns. A NAPES developer specifies the state machines (i.e., states and state transitions) for each of an application's components.

State transitions occur in response to events, which may originate locally (timer events) or may arrive from other components of the application. Thus, the events serve the purpose of intra- and inter-component coordination.

Besides lightweight event-based coordination, components communicate via flows, which form the traffic stress-testing the network under investigation. A component may generate a flow addressed to another component. A flow is a sequence of packets described with some parameters, e.g., ones that describe a distribution of inter-packet times and a distribution of packet sizes. Flows are sent and received by means of components' ports. A client port generates the packets of a flow, while a server port receives the packets.

The generation of flows is governed by the states of the states machines inside the components comprising a NAPES application. Specifically, for each client port, a NAPES user specifies a flow to be generated in each state of a state machine assigned to control the client port (in some states there may be no flow at all). Thus, when an event (a local one or received from another component) causes a state transition, flows generated by the component's client ports controlled by the state machine in question may change. As a result of state changes of components' state machines, a NAPES application may generate traffic of different patterns. Overall, the above mechanisms allow a NAPES to approximate some actual IoT application, whose components are also likely "to be in different states" reflecting the state of the environment.

In terms of implementation, a component is specified in a data structure to be interpreted by the so-called NAPES Runtime. A NAPES Runtime should be installed on a computing device to make it a NAPES node. We intend to implement a NAPES Runtime on different platforms, from resource-constrained IoT nodes to cloud servers. We currently have a Linux implementation; runtimes for Android and Arduino are being developed. As the accepted component data structure is the same, no matter what the underlying platform, NAPES components are portable and can be run on different nodes. Inter-component events are exchanged as short Message Queuing Telemetry Transport (MQTT) messages.

A general approach to stress-test a network with NAPES is to (a) develop a multi-component NAPES application with an application logic close to that of an actual application, (b) connect to the network under test a number of NAPES nodes (computing devices with NAPES Runtimes), (c) distribute the data structure representations of the application's components to the NAPES nodes, (d) run the application and collect logs. Notably, NAPES applications are agnostic as to an underlying stress-tested network. There is no application-to-network signaling, which calls for some network intelligence and adaptivity.

4. Experiments and Results

The AI algorithms were tested with the network topology shown in Figure 2. The test network consists of thirteen NAPES hosts on which a simple client server application is deployed. All network's links have the capacity of 50 Mbps and the access links (between hosts and network switches) have the capacity of 1 Gbps. The hosts at the top of Figure 2 represent sensor nodes (e.g., equipped with cameras) and the host at the bottom represents a cloud server. The network topology was setup in such a way that it becomes easily congested around the cloud server node, when shortest path routing is used, while the network has still enough capacity to handle the traffic generated by the active sensors.

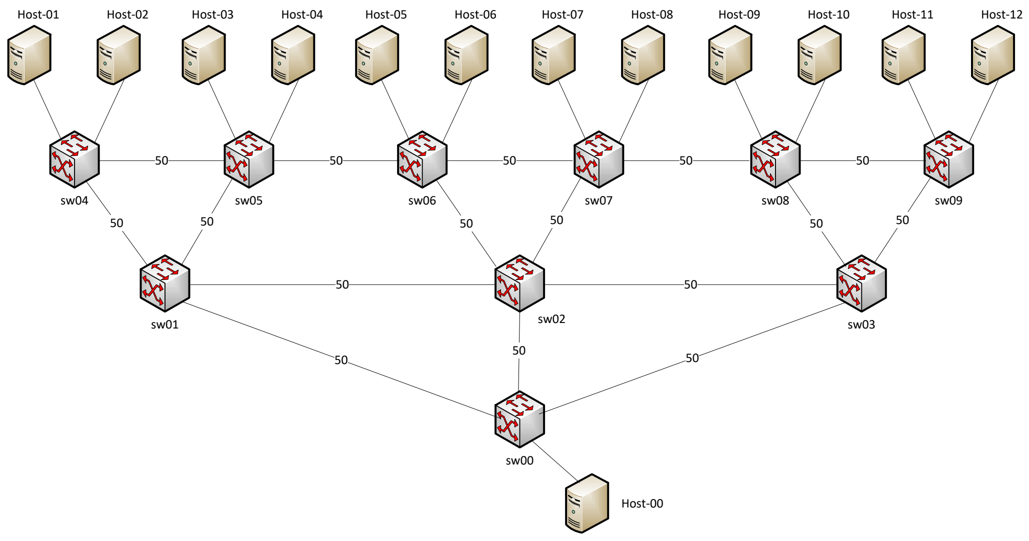

Figure 2. Analyzed network topology.

The application emulates the case when a space (e.g., the streets of a smart city) is instrumented with multiple sensor nodes. A sensor node may be in the standby or alert mode. When in the standby mode, no traffic is generated; when in the alert mode, a node generates a simple User Datagram Protocol (UDP) flow with the specific bit rate of 20 Mbps. Note that the bit rates have been chosen to allow convenient experimentation with the AI algorithm; they need not be realistic from the application domain point of view. Imagine that some stimulus (e.g., noise from a street sweeper vehicle at night time) triggers a change from the usual standby mode to the alert mode. As the sweeper moves down the streets, different sensor nodes are exposed to the noise and enter the alert mode (starting generating traffic).

We assumed that two sensor nodes enter the alert mode at a time (imagine two cameras deployed at each intersection). Moreover, pairs of sensors are alerted according to a regular pattern. Specifically, NAPES components running on the nodes Host-01 and Host-02 start generating traffic at the time 0 s (one flow per node), components on the nodes Host-03 and Host-04 start at the time 20 s, components on the nodes Host-05 and Host-06 start at the time 40 s, etc. Each component remains in the alert mode for 40 s and, afterwards, returns to the standby mode (with no traffic generated), which lasts for 100 s. After that, a component enters the alert mode again.

To conduct the experiments with the AI algorithm, we emulated the above test network using the MiniNet tool [35]. The network nodes represent the Open Virtual Switch (OVS). The test flows were generated using the NAPES traffic generator. In all cases the test instances were run on the hardware platform with the following parameters:

- 16 GB RAM
- 8 VCPU
- 64 GB of disk space
- Ubuntu system version 20.04.

In the following experiments, we compared 2 network parameters i.e., total throughput obtained by the node representing cloud server and temporary network loss rate (averaged over all flows) to show the advantage of the proposed AI algorithm.

Figure 3 shows total throughput without AI and Figure 4 shows total throughput with the AI algorithm. It is noticeable that the AI algorithm allows for better network resource utilization by rerouting traffic to alternative paths and as a result increases the overall network throughput. The hosts Host-01 ... Host-12 are arranged in pairs that periodically switch between active and inactive states. When four hosts become active at the same the network becomes congested if all flows are routed over the same link (this may happen with shortest path routing). With the AI algorithm, the flows can be routed over link disjoint paths, thus, network congestion can be avoided. This can be observed, for example, in the time period between 20 and 40 s. Without AI, the server throughput is only 50 Mbps, as all active flows are routed over a single link with a capacity of 50 Mbps. When the AI algorithm is enabled, two flows can be rerouted to alternative disjoint paths and the throughput increases to 80 Mbps (which is equal to the offered traffic).

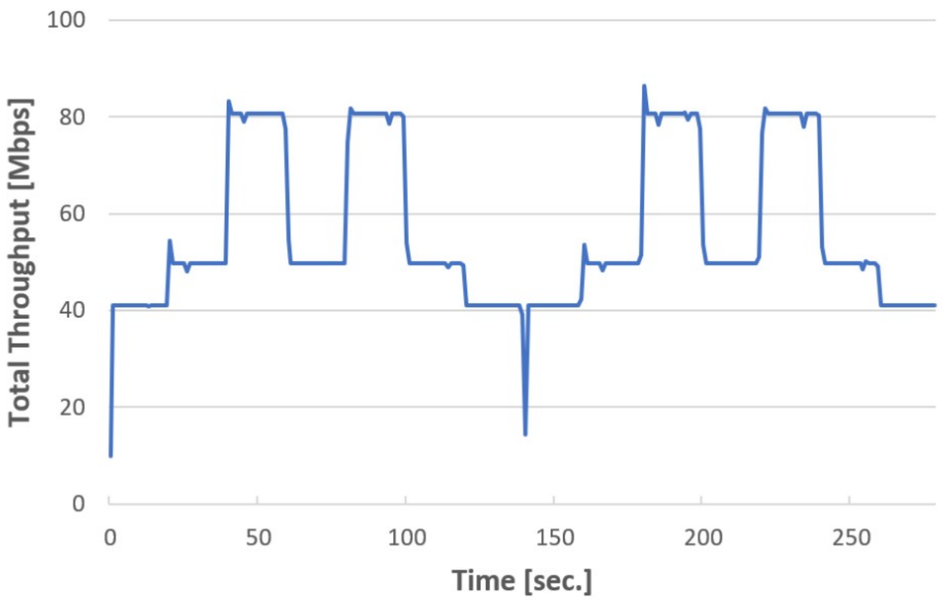

Figure 3. Total throughput without AI.

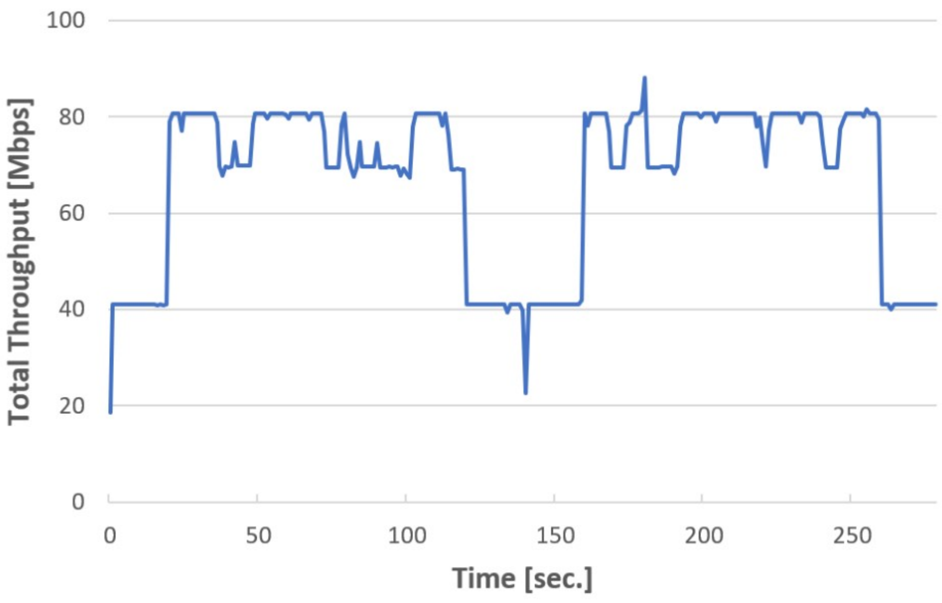

Figure 4. Total throughput with AI.

The advantage of using AI is even clearer for the loss rate parameter. With the AI algorithm, network losses are significantly lower than without using AI (compare Figures 5 and 6). However, the traffic losses cannot be completely avoided as the AI algorithm needs some time to “observe” the traffic prior to the path switch over decision.

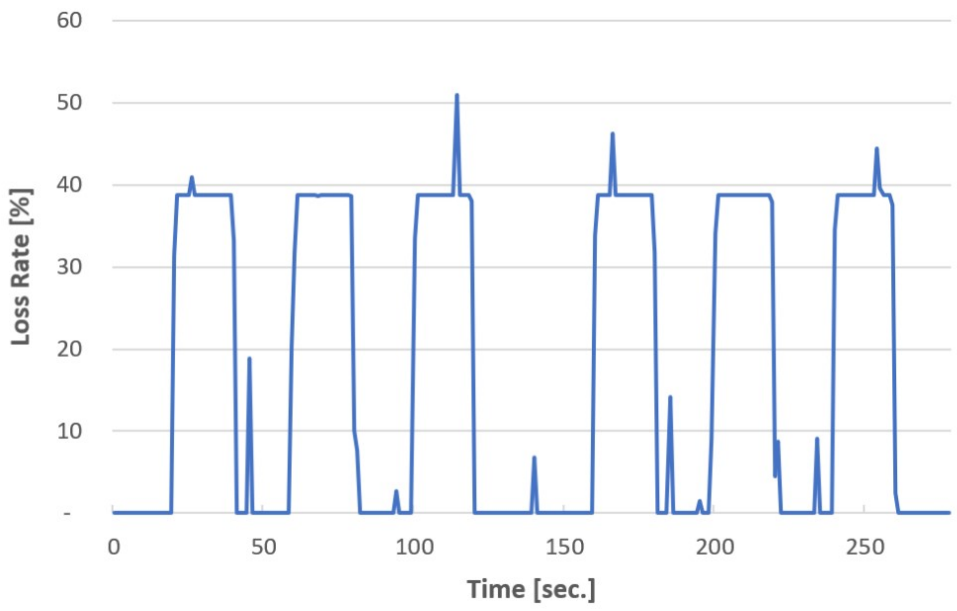

Figure 5. Loss rate without AI.

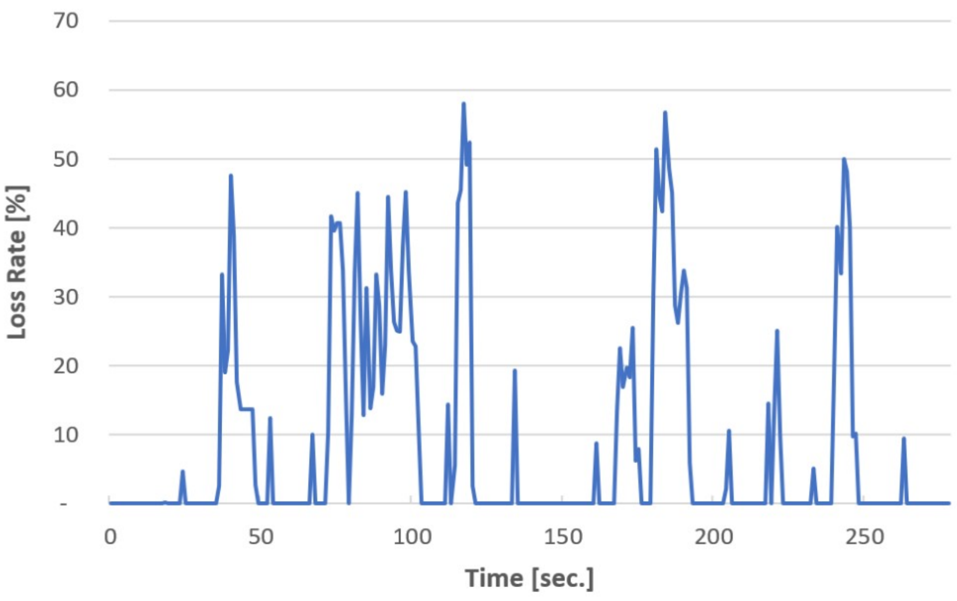

Figure 6. Loss rate with AI.

It can be seen in Figure 5 that the network suffers from six congestion periods due to the naive shortest path routing (the same can be seen also in in Figure 3). It happens when four hosts nearest to the same aggregation switch (sw01, sw02, and sw03) are all operating. The situation happens two times for each aggregation switch during the 280 s experiment. The first congestion period (form 20 to 40 s) happens when hosts: Host-01, Host-02, Host-03, and Host-04 are operating and all the traffic is routed on the shortest path. In such a case, link sw00-sw01 becomes the bottleneck, because all the traffic is using it. However, in the presented AI framework, each intent can chose between two paths that are as link disjointed as possible. It is clearly seen in Figure 2 that, for each host (Host01-Host-12) in the considered network, there exists a pair of paths to the server (Host-00) that shares only two links. These links are directly connected either to the host or to the server. Therefore, there is always a combination of possible paths, which can be assigned to intents, that will prevent from the described congestion situations on the links connecting the server with the aggregation switches. As displayed in Figure 4, the presented AI framework usually finds this perfect assignment and allows for considerable loss reductions.

The results of the above experiments obtained for five different test cases are presented in Tables 1 and 2. Table 1 shows the per flow average throughput obtained for experiments performed with and without the AI algorithm. Table 2 shows similar results for the average flow loss rate parameter. We can see that the traffic losses (and the resulting throughput reductions) are not evenly distributed between flows. Without the AI algorithm, the per flow loss rate can be as high as 40 % (meaning the same level of throughput decrease). The AI algorithm reduces the average per flow loss rate over 3 times, to an average level of 6 % (from an average of 18 % for the case with the shortest path routing).

Table 1. Comparison of flows' throughput for different test case.

(a) Throughput without AI

Flow No.	Test Case					Average
	1	**2**	**3**	**4**	**5**	
1	3.58	5.75	3.68	4.83	5.23	4.61
2	5.75	4.97	5.52	4.50	3.72	4.89
3	5.21	3.61	4.11	4.60	3.98	4.30
4	4.01	4.34	5.30	4.71	5.64	4.80
5	4.26	4.24	5.52	5.62	3.74	4.68
6	4.31	5.69	4.25	4.50	5.42	4.83
7	5.59	4.95	4.61	4.79	5.39	5.07
8	4.52	3.83	4.33	3.78	4.16	4.12
9	5.24	5.50	5.55	5.42	4.17	5.18
10	3.81	4.76	4.16	3.79	4.46	4.20
11	5.69	4.76	5.00	4.64	4.46	4.91
12	3.98	3.68	3.99	4.85	5.62	4.42

(b) Throughput with AI

Flow No.	Test Case					Average
	1	**2**	**3**	**4**	**5**	
1	5.69	5.43	5.54	5.59	5.58	5.57
2	5.65	5.77	5.14	5.77	5.66	5.60
3	5.75	5.33	4.66	5.71	5.07	5.30
4	5.46	5.71	5.76	5.58	5.74	5.65
5	5.52	4.66	5.19	4.98	5.57	5.19
6	5.19	5.72	5.42	4.97	4.61	5.18
7	5.37	5.59	5.69	5.52	5.03	5.44
8	5.68	5.60	5.31	5.74	5.55	5.58
9	5.56	5.70	5.30	5.13	5.24	5.39
10	5.31	4.97	4.96	5.03	5.68	5.19
11	5.65	5.44	5.76	5.65	5.72	5.64
12	5.61	5.56	5.59	5.45	5.77	5.60

Table 2. Comparison of flows' loses for different test case.

(a) Loss rate without AI

Flow No.	**Test Case**					**Average**
	1	**2**	**3**	**4**	**5**	
1	38.10	0.57	36.41	16.40	9.46	20.19
2	0.49	14.09	4.57	22.11	35.65	15.38
3	9.85	37.60	28.85	20.44	31.19	25.59
4	30.58	25.00	8.36	18.45	2.40	16.96
5	26.18	26.62	4.40	2.65	35.22	19.01
6	25.32	1.55	26.48	22.14	6.26	16.35
7	3.26	14.36	20.22	17.21	6.81	12.37
8	21.81	33.83	25.16	34.54	28.09	28.69
9	9.39	4.76	3.94	6.23	27.89	10.44
10	33.93	17.70	28.00	34.34	22.76	27.35
11	1.81	17.67	13.38	19.84	22.88	15.12
12	31.19	36.31	31.03	16.16	2.75	23.49

(b) Loss rate with AI

Flow No.	**Test Case**					**Average**
	1	**2**	**3**	**4**	**5**	
1	1.49	5.98	4.08	3.33	3.57	3.69
2	2.29	0.24	11.14	0.15	2.07	3.18
3	0.58	7.79	19.35	1.23	12.34	8.26
4	5.63	1.18	0.38	3.40	0.65	2.25
5	4.47	19.34	10.17	13.74	3.52	10.25
6	10.13	1.07	6.17	13.91	20.23	10.30
7	7.12	3.26	1.55	4.56	13.07	5.91
8	1.73	3.22	8.11	0.66	4.04	3.55
9	3.71	1.35	8.34	11.21	9.27	6.77
10	8.03	13.95	14.24	12.91	1.72	10.17
11	2.24	5.86	0.48	2.37	1.15	2.42
12	2.98	3.80	3.33	5.67	0.20	3.20

5. Conclusions

In this paper, the management of SDN network resources with AI support for IoT applications was presented. Particularly, the studied problem focuses on an optimal switching of intents between two available paths in an SDN networks using AI algorithm. In the developed architecture, the switching is performed based on Deep-Q-Network AI mechanism where its main purpose is to maximize the total throughput in the SDN network with minimal average total data loss rate for served IoT applications. The presented approach is an intelligent SDN management system with AI support, especially applicable in programmable next-generation networks (5G and beyond).

Moreover, the network resource allocation in context of QoS assurance is a growing problem, especially for IoT services. The designed solution is suitable for fast and optimal resource allocation, especially in cases of emergency and delay-sensitive IoT services. To test the proposed AI solution, we used an IoT traffic generator called NAPES, developed within the FlexNet project. We compared the performance and quality parameters, i.e., total throughput in the network and average data loss rate, to evaluate the AI usefulness in the designed solution. The presented results confirm effectiveness of proposed AI approach, which significantly improves the overall Quality of Service and network performance. In particular, it maximizes the total network throughput as well as significantly reduces the average total data loss rate. The obtained results confirm the rightness of applying AI to the presented problem.

Further research will be focused on more comprehensive experiments that include real scenarios from IoT use cases with more extensive network topologies, with different QoS parameters (losses and latency) and with real traffic scenarios from IoT vertical services, loss and delay sensitive.

Author Contributions: Conceptualization, M.Ż. and S.K.; methodology, M.Ż., S.K., A.B.. J.D. and Z.K; software, M.Ż., J.D., A.B. and W.S.; validation, S.K., Z.K. and A.B.; formal analysis, M.Ż. and Z.K.; data curation, A.B., S.K. and Z.K.; writing—original draft preparation, S.K., M.Ż., J.D., Z.K. and A.B.; writing—review and editing, S.K., M.Ż., J.D. and Z.K.; supervision, Z.K. and S.K.; project administration, Z.K. and S.K.; funding acquisition, Z.K. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was funded by the FlexNet project in EUREKA CELTIC-NEXT Cluster for next-generation communications together with The National Centre for Research and Development in Poland.

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

Abbreviations

The following abbreviations are used in this manuscript:

QoS	Quality of Service
KPI	Key Performance Indicator
IoT	Internet of Things
SDN	Software-Defined Networking
AI	Artificial Intelligence
NAPES	Networked Application Emulation System
5G	Network of 5th Generation
FlexNet	Flexible Network
ML	Machine Learning
KDN	Knowledge-Defined Network
ONOS	Open Network Operating System
SAT	Satisfiability Problem
IMR	Intent Monitor and Reroute service
MQTT	Message Queuing Telemetry Transport (standard messaging protocol for IoT)
Gbps	Giga bit per seconds
UDP	User Datagram Protocol
OVS	Open Virtual Switch

References

1. Liyanage, M.; Ylianttila, M.; Gurtov, A. Securing the Control Channel of Software-Defined Mobile Networks. In Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, Sydney, NSW, Australia, 19 June 2014. [CrossRef]
2. Rothenberg, C.E.; Nascimento, M.R.; Salvador, M.R.; Corrêa, C.N.A.; Cunha de Lucena, S.; Raszuk, R. Revisiting Routing Control Platforms with the Eyes and Muscles of Software-Defined Networking. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, Helsinki, Finland, 13 August 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 13–18. [CrossRef]
3. Kreutz, D.; Ramos, F.M.V.; Verissimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
4. IETF. *Software-Defined Networking: A Perspective From Within a Service Provider Environment*; IETF: Fremont, CA, USA, 2017.
5. de la Oliva, A.; Li, X.; Costa-Perez, X.; Bernardos, C.; Bertin, P.; Iovanna, P.; Deiss, T.; Mangues-Bafalluy, J.; Mourad, A.; Casetti, C.; et al. 5G-TRANSFORMER: Slicing and orchestrating transport networks for industry verticals. *IEEE Commun. Mag.* **2018**, *56*, 78–84. [CrossRef]
6. Dinh, K.T.; Kukliński, S.; Osiński, T.; Wytrębowicz, J. Heuristic traffic engineering for SDN. *J. Inf. Telecommun.* **2020**, *4*, 251–266. [CrossRef]
7. Bera, S.; Misra, S.; Vasilakos, A.V. Software-Defined Networking for Internet of Things: A Survey. *IEEE Internet Things J.* **2017**, *4*, 1994–2008. [CrossRef]
8. Municio, E.; Marquez-Barja, J.; Latre, S.; Vissicchio, S. Whisper: Programmable and Flexible Control on Industrial IoT Networks. *Sensors* **2018**, *18*, 4048. [CrossRef] [PubMed]

9. Municio, E.; Latre, S.; Marquez-Barja, J.M. Extending Network Programmability to the Things Overlay Using Distributed Industrial IoT Protocols. *IEEE Trans. Ind. Inform.* **2021**, *17*, 251–259. [CrossRef]
10. Zemrane, H.; Baddi, Y.; Hasbi, A. SDN-Based Solutions to Improve IOT: Survey. In Proceedings of the 2018 IEEE 5th International Congress on Information Science and Technology (CiSt), Marrakech, Morocco, 21–27 October 2018; pp. 588–593. [CrossRef]
11. Rego, A.; Canovas, A.; Jiménez, J.M.; Lloret, J. An Intelligent System for Video Surveillance in IoT Environments. *IEEE Access* **2018**, *6*, 31580–31598. [CrossRef]
12. Omar, H. Intelligent Traffic Information System Based on Integration of Internet of Things and Agent Technology. *Int. J. Adv. Comput. Sci. Appl.* **2015**, *6*, 37–43. [CrossRef]
13. Jin, Y.; Gormus, S.; Kulkarni, P.; Sooriyabandara, M. Content Centric Routing in IoT Networks and Its Integration in RPL. *Comput. Commun.* **2016**, *89*, 87–104. [CrossRef]
14. Flexnet. Flexible IoT Networks for Value Creators. 2020. Available online: www.celticnext.eu/project-flexnet (accessed on 22 September 2021).
15. Choque, J.; Aguero, R.; Kopertowski, Z.; Nguyen, K.K.; Medela, A.; Municio, E.; Marquez-Barja, J.M.; Domaszewicz, J.; Bak, A.; Lee, J.H.; et al. FLEXNET: Flexible Networks for IoT based services. In Proceedings of the 2020 23rd International Symposium on Wireless Personal Multimedia Communications (WPMC), Okayama, Japan, 19–26 October 2020; pp. 1–6. [CrossRef]
16. Kozdrowski, S.; Banaszek, M.; Jedrzejczak, B.; Żotkiewicz, M.; Kopertowski, Z. Application of the Ant Colony Algorithm for Routing in Next Generation Programmable Networks. In *Computational Science–ICCS 2021*; Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 526–539.
17. Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. *White Paper* **2012**.
18. Zhao, Y.; Le, 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]
19. Mao, H.; Alizadeh, M.; Menache, I.; Kandula, S. Resource Management with Deep Reinforcement Learning. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks, HotNets '16, Atlanta, GA, USA, 9–10 November 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 50–56. [CrossRef]
20. Kozdrowski, S.; Cichosz, P.; Paziewski, P.; Sujecki, S. Machine Learning Algorithms for Prediction of the Quality of Transmission in Optical Networks. *Entropy* **2021**, *23*, 7. [CrossRef] [PubMed]
21. Chen, B.; Wan, J.; Lan, Y.; Imran, M.; Li, D.; Guizani, N. Improving Cognitive Ability of Edge Intelligent IIoT through Machine Learning. *IEEE Netw.* **2019**, *33*, 61–67. [CrossRef]
22. Abar, T.; Letaifa, A.; 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. [CrossRef]
23. Dobrijevic, O.; Santl, M.; Matijasevic, M. Ant colony optimization for QoE-centric flow routing in software-defined networks. In Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM), Barcelona, Spain, 9–13 November 2015; pp. 274–278. [CrossRef]
24. Ali, J.; Roh, B.H. Management of Software-Defined Networking Powered by Artificial Intelligence. 2021. [CrossRef]
25. Mishra, P.; Puthal, D.; Tiwary, M.; Mohanty, S.P. Software Defined IoT Systems: Properties, State of the Art, and Future Research. *IEEE Wirel. Commun.* **2019**, *26*, 64–71. [CrossRef]
26. 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. *SIGCOMM Comput. Commun. Rev.* **2017**, *47*, 2–10. [CrossRef]
27. Sanvito, D.; Moro, D.; Gullì, M.; Filippini, I.; Capone, A.; Campanella, A. ONOS Intent Monitor and Reroute service: Enabling plug amp;play routing logic. In Proceedings of the 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), Montreal, QC, Canada, 25–29 June 2018; pp. 272–276. [CrossRef]
28. Karp, R. Reducibility among combinatorial problems. In *Complexity of Computer Computations*; Miller, R., Thatcher, J., Eds.; Plenum Press: New York, NY, USA, 1972; pp. 85–103.
29. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
30. Liang, E.; Liaw, R.; Nishihara, R.; Moritz, P.; Fox, R.; Goldberg, K.; Gonzalez, J.; Jordan, M.; Stoica, I. RLlib: Abstractions for Distributed Reinforcement Learning. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; Dy, J., Krause, A., Eds.; PMLR, Proceedings of Machine Learning Research: Stockholm, Sweden, 2018, Volume 80, pp. 3053–3062.
31. Ruffy, F.; Przystupa, M.; Beschastnikh, I. Iroko: A Framework to Prototype Reinforcement Learning for Data Center Traffic Control. *arXiv* **2018**, arXiv:1812.09975.
32. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. OpenAI Gym. *arXiv* **2016**, arXiv:cs.LG/1606.01540.
33. Marques, E. Goben. 2018. Available online: https://github.com/udhos/goben (accessed on 22 September 2021).

34. Kingma, D.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the International Conference on Learning Representations, Banff, AB, Canada, 14–16 April 2014.
35. Xiang, Z.; Seeling, P. Chapter 11-Mininet: An instant virtual network on your computer. In *Computing in Communication Networks*; Fitzek, F.H., Granelli, F., Seeling, P., Eds.; Academic Press: Cambridge, MA, USA, 2020; pp. 219–230. [CrossRef]