NLP4: An Architecture for Intent-Driven Data Plane Programmability

NLP4: An Architecture for Intent-Driven Data Plane Programmability

Antonino Angi * Alessio Sacco * Flavio Esposito ‡ Guido Marchetto * Alexander Clemm †
* Department of Control and Computer Engineering, Politecnico di Torino, Italy
‡ Computer Science Department, Saint Louis University, USA
† Futurewei Inc., USA

***Abstract*—Translating high-level policies to lower-level network rules is one of the main goals of control or data plane network programmability. To further abstract requirements and propel automation in networking, several industries have proposed the paradigm of "network intent". However, the translation from intents to low-level policies is considered critical to program data planes and other network elements, especially when dealing with P4-enabled switches. In this paper, we present NLP4, an architecture that helps translate intents, in the form of human language, into data-plane programs, in the form of P4 rules. In particular, NLP4 uses Natural Language Processing (NLP) techniques to translate high-level human-language intents, a MultiLayer Perceptron (MLP) model for processing the NLP output and converting it into mid-level policy. An API then uses this information, which separates the intent from the network to generate commands readable by P4-enabled switches. Our initial prototype on a network emulator validates our architecture for a specific case: load profiling, demonstrating how even users with limited P4 expertise may customize their networks by merely specifying intents.**

***Index Terms*—network intent, load profiling, machine learning**

I. Introduction

Trying to make our networks more programmable has been a goal of the networking research and business community for the last few years [1]. Recent advances in data-plane programmability have allowed implementing customized high-speed network services directly into programmable switches. Programming Protocol-independent Packet Processors (P4) is an "open source, domain-specific programming language for network devices, specifying how data plane devices (switches, routers, NICs, filters, etc.) process packets" [2]. P4 language allows network operators to realize custom network functions without tailoring them to the specific networking hardware in use.

The need for data plane intent programmability abstractions. While P4 has been promising as it lets operators customize network functions at a line rate, writing functional P4 programs still arguably requires detailed knowledge and a steep learning curve, even for networking experts.

The main focus of researchers in programming languages, in general, and network programming languages, in particular, has been on either making programming safer [3]–[5] or easier [6]. Many successful studies have been conducted to reach these goals [7], [8]. Some of them focus on control plane programmability, such as OpenFlow [9], others on making easier or more elastic data plane programs [6], [10].

One of the aims of intent-driven networking is precisely this: simplify network requirement specification, making network programming accessible to inexperienced users. Others researchers instead proposed applying natural language processing for programming in general or network intents, as shown in this recent survey [11]. A particularly inspiring approach was proposed by Riftadi and Kuipers [12]. Their solution uses an Intent Definition Language specifically for intent interpretation and translation.

Our contribution. Inspired by their solution, we are taking their approach one step further by applying Natural Language Processing to close the loop. In particular, our solution, NLP4, translates a specific input given by the user to a P4 program so that the user can customize the network as he wants, following the P4 criteria. This is done using a combination of mechanisms, as shown in Figure 1. First, a user or a program generates the intent; such intent is then preprocessed returning an array that is tokenized and encoded using a dictionary of words. Subsequently, we apply a MultiLayer Perceptron (MLP) model to obtain an array where each cell corresponds to the element of the network the intent is referred to. For example, it can be a server, if the intent is to operate on servers traffic, or ports of the switch, if the intent operates on the link load. Independently of the network element, such mapping gives information on the main goal, and to achieve this user goal we modify the behavior of main network forwarding elements. To do so, NLP4 contains an API used to convert this encoded array into the P4-enabled switches' configuration files.

We then evaluate our approach with an initial prototype tested over a virtual network testbed. The focus of our evaluation has been on a load profiling [13] use case, a superset of load balancing. In particular, we evaluate how P4 switches can be automatically programmed to meet the desired profile of traffic expressed by the user.

The rest of the paper is structured as follows. Section II presents other state-of-the-art solutions; in Section III we give an overview on NLP4's architecture. Possible application examples are shown in Section IV. Our results are presented and described in Section V. Finally, we conclude the paper in Section VI.

The work of Antonino Angi and Alessio Sacco was performed in the Department of Computer Science at Saint Louis University.

II. Related work

Many studies have tried to automatize the network as much as possible, proposing solutions that combine frameworks to low-level policy translator [14], [15]. Some of them attempt to improve the expressiveness of data-plane programming languages and facilitate customization and simulation on different hardware vendors.

For example, Pyretic is a Python-based language that abstracts the possibly complicated network rules [16]. Another language, Merlin [17], was deployed with the goal of expressing networking rules as a series of logical predicates to manage traffic among the network. P4 is another recent programming language for data-plane programmable switches, which soon became one of the most used [2]. However, experience with P4 programming has shown that it is not an easy programming language, not providing enough abstract features. Many attempts were made to make P4 easy writable; although it is still challenging to write code with this programming language as it is still very low-level [6]. Beside, these languages are more focused on managing network administration issues rather than identifying, understanding, and translating application requirements.

To simplify coding in P4, researchers started using intents and, in particular, intent-driven networking techniques, allowing even inexperienced user to customize their networks, such as Nile [18] and Marple [19]. Nile uses the human language to retrieve the intent and get the feedback of the retrieved text to improve other translations. Marple uses P4 to perform network monitoring and performances evaluation, translating dynamic queries into primitives where the result is stored. Despite being very successful and helpful for future studies, these works still do not help create the topology used by P4 at startup.

Recently organizations such as IETF and ONF have been working hard to provide a NorthBound Interface (NBI) suitable for intent and applicable to SDN contexts [20]. Researches have also added recent software techniques to provide capacities of expressing intents more easily (e.g., human language); see for example [14]. In particular, this work focuses on applying a Behaviour Driven Development framework in Python, *behave*, and combines it with intent-policy translators and interpreters. However, this work does not allow to customize a switch's forwarding rules using data-plane programming languages. Other recent solutions apply Machine Learning (ML) techniques to abstract low-level network details from a natural language and, specifically, from a text written in English [21], [22].

Differently from previous work, we apply an Intent-Driven API to interpret the user needs by converting those into P4 configuration files applied directly to each switch of the network, customizing their forwarding rules. This is possible using Natural Language Processing and MultiLayer Perceptron techniques, and we show how, thanks to the combination of these mechanisms, even a non-experienced user can customize a data plane for its network without having to code in P4.

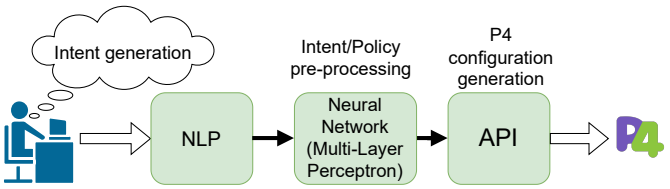

Fig. 1: NLP4: Overview and Workflow. The user intent is first preprocessed and converted into an intermediate representation. Then, based on this, our API generates the commands for P4 switches.

III. NLP4 Architecture Overview

This section describes the main components of NLP4 focusing on its design particularities. As represented in Figure 1, our solution is composed of three components operating sequentially: *(i)* an NLP technique using text mining for preprocessing purposes, *(ii)* an MLP model to process the output from this first phase, *(iii)* and an API to finally add the intent specification in the configuration files used by P4 switches. In particular, after the user has specified the intent and desired behavior of the network, the NLP model processes the natural language to obtain a compact representation that expresses the context. This information is then used by the MLP to generate the mid-level policy that is retrieved by our API, responsible for populating the configuration file accepted by the P4 switches when they startup.

A. Interpreting User Language via NLP

When a user specifies the intent, it can be done in English or other natural languages as our solution is not language-dependent. However, this intent might contain words not beneficial for the general understanding that could increase the computational time to retrieve the meaning of the phrase or, even worse, could bring to a misunderstanding of the whole sentence. For this purpose in NLP4, we use Natural Language Processing (NLP) techniques as the first part of our process: the preprocessing part.

NLP is a branch of Artificial Intelligence (AI) whose aim is to understand the natural language and extract the main information in order to perform activities such as content analysis, chatbot, and sentiment analysis, but even preprocessing tasks. NLP offers two main functionalities: Natural Language Understanding (NLU) and Natural Language Generation (NLG). The former is used to convert the input into an easily analyzable representation. Therefore NLU only focuses on catching the meaning of the human language rather than processing it. The latter generates text from various input representations. Such a representation might be numerical or even visual input data [23].

During this preprocessing phase, we first go into a text cleaning function, which better organizes and structures our intents, deleting potentially unnecessary words and punctuation that could bring noise to our ML model, and returning a list of tokens. Secondly, we transform our tokens into their root form using normalizing rules.

In literature, there are alternative normalization techniques to accomplish this goal: lemmatization and stemming. The first uses vocabulary to reduce words to their dictionary form [24]; the second brings all words characterized by the same root (stem) to their common form. A recent study [25] shows that stemming performs better and gives better results with short sentences. As for our intent-based network, we are not expecting to have long queries as inputs; therefore, we decide to go further with the stemming techniques, and especially with the Porter' stemmer, which is known nowadays as one of the most used [24].

Lastly, since machines do not understand human language, the NLP model needs to convert this knowledge into a numerical representation. As such, it counts the words appearing on the intent, tokenizes them, builds a dictionary, and generates the final output in the form of an encoded vector. This vector will be the input of the subsequent MLP module.

B. MLP for Mid-level Policy Generation

A MultiLayer Perceptron (MLP) is an artificial Neural Network (NN) composed of one or more hidden layers between the input and the output ones. Particularly, an MLP is a model of NN which is fully connected, does not contain any loop between nodes (i.e., is a feed-forward NN) and learns via the Back-propagation approach. The main reason for using an MLP model is to find sufficient parameters and good generalization for classification or regression tasks because of its flexibility when applied to different contexts and types of data [26]. In an MLP model, the input layer is composed of many neurons as the number of measurements for the prediction model; meanwhile, the number of neurons in the output layer equals the number of classes [27].

In our prototype, we set the MLP model with two hidden layers: the first consists of 20 neurons (whose value is achieved after a preliminary performance analysis) and uses the rectified linear unit activation function; the second is characterized as many neurons as the number of the network elements, i.e., servers or switches, of the topology, and uses the sigmoid activation function.

The output of this MLP model is thus a mid-level policy that encodes the action to the involved network elements. Finally, these values are packed in a single vector and passed as input to our Intent-Driven API.

C. Intent-Driven API

As the next and final step of our workflow, we use an Application Programming Interface (API) to add the output from the MLP model to the configuration files used by P4 at startup.

Our API is written and follows the idea of *behave*, a Behaviour Driven Development (BDD) framework. BDD is an agile software technique that allows people with less technical experience to specify some high-level requirements, avoiding further low-level details on how to reach them. This can be done by defining the needed specifics with a special language, called Gherkin, which uses the *Scenario-Given-When-Then* (SGWT) formula. BDD finds applicability for various purposes [28], such as testing verification techniques for software-defined networking behavior, despite its maintenance cost [29].

In NLP4, we use an Intent-Driven API in Python to convert the the intent specification, which has already been processed and analyzed by the NLP and MLP components, to a configuration file for the P4 switches. We limit our focus to the switch behavior so that, even when the intent is servers specific, we configure the switches accordingly. For example, if the user wants to disable a server, our solution would interact with the switches by putting a weight of 0 to the port associated to that server. In such a way, our approach with this API is shown to be easily adaptable for users with minimal experience in network management and coding.

Although P4 switches can be used for many purposes aside from the simple forwarding, in this paper we limit the possible P4 actions to a load profiling use case [13], [30], leaving the implementation of other actions as future work. In this load profiling scenario, we allow the user to specify how to split traffic over outgoing links of a switch and the desired load on these links. If links have different features or traffic have different priorities, an unequal balance may be convenient, where these balances constitute the profiles. Our P4-enabled switches are thus programmed to read these profiles from the configuration file and implement the desired policy.

D. P4-enabled Switches

For this solution, we used P4 programmable switches as it is one of the most used programming languages that allows customizing data-planes. However, the P4 syntax is quite elaborate, and we argue that an intent-based approach to program the switches can open new business opportunities and new research activities.

P4 is composed of three main blocks: a parser, a match-action pipeline, and a deparser. The parser is structured as a finite-state machine used first to analyze and extract headers. A packet, for example, could start with an Ethernet header, followed by either IPv4 or IPv6 header; the parser extracts all these headers, passing them to the match-action pipeline. The match-action pipeline contains tables that P4 manipulates to customize the switch behavior and perform routing strategies according to the defined policy. The deparser is the last stage of our P4 runtime, and it is used to collect the potentially modified headers that have been through the match-action pipeline.

It is well known that writing code in P4 is challenging, especially for newbies. Since its commands must be executed at line rate, P4 has limitations in the possible instructions. For example, it does not allow loops cycles, and the code has to be developed using switch-case and if-else cases. Although developed with the idea of increasing abstraction from an even lower-level language such as direct writing code for PISA architecture, P4 is considered as a complex language due to the limited available resources, the inability to use loops cycles, and the little feedback that the compiler gives when there are errors. This brings the programmer to make various trials and

errors, being guided by log files to understand how to fix the code, which is an incredibly time-consuming process for the programmer.

There have been steps to make P4 more abstract and easier programmable, using shared modules in various applications and switch specifications hardware. However, we are far from considering P4 as a high-level language that newbies can easily learn with no much effort [6]. There have also been studies that enable users and applications to give requirements in the form of intents and have them translated to a more low-level language for data-plane programmable switches [12].

In NLP4, we modified our P4 code according to the intent specified by the user. To do so, we stored the information contained on the switches' configuration files into registers that are then retrieved at P4 startup. We wrote the code according to P4 rules in which all the loop cycles were replaced with a cascading series of if-cases, and the register could have only been manipulated inside match-action tables.

Furthermore, when implementing the load profiling use case, we created multiple match-action tables in order to manipulate the registers that contained the intent specification. This is due to P4 expressiveness limitations, which do not allow a register to be manipulated inside if-else cases and we have dealt the problem by moving the manipulation inside apply modules when there is a match inside a table. This simple example of P4 complexity further motivates an intent-based architecture as the one proposed in this paper.

IV. Illustrative Examples

To explain the functionalities of NLP4 in general and of each component, in this section we present two simple use cases and the steps performed by NLP4 when dealing with them.

A. Load Profiling

A possible intent specification is load profiling, in which NLP4 is used to adapting the P4 switch behavior to network operator demanded profiles. However, not necessarily the user must specify the profiles, but he/she can express some objectives. For example, a possible high-level intent can be "Disable server 2 because of maintenance". After the NLP preprocessing we would get a dictionary of words:

```
{'disabl': 1, 'server': 2,
'2': 3, 'mainten': 4}.
```

Subsequently, we applied our MultiLayer Perceptron (MLP) model to get a mid-level policy corresponding to the specified intent. Our MLP model would return an array corresponding to the elements on the network interested by the intent. In our case, considering a topology composed of 4 servers, the output would be: "[1 0 1 0 0]", where the first value indicates the intent main goal (presuming that "1" is linked to "disable" in our dictionary), and the other values correspond to the server which had to be applied. This specific vector is then used by our API to create the configuration files for all switches in the network to redirect traffic to other servers, while the switch attached to S1 would block that port. While in this case the vector is referred to the servers, this approach can be generalized to all elements on the network (i.e. switches, servers, ports) so that it can easily reproduce the intent specified by the user. The API detects the specific element using the first value, and then configure the switches accordingly.

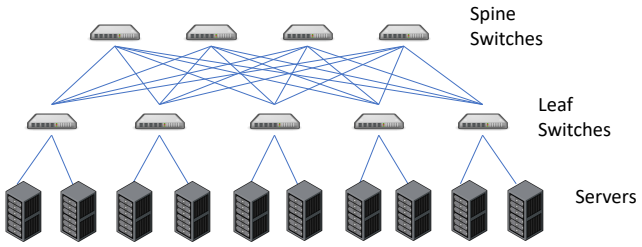

Fig. 2: Network topology used in the experiments.

B. Traffic priority

NLP4 can be also used to give priority to a certain traffic coming from or going to a specified server. For example, if the user wants to give priority to traffic on the network going to/from a server, e.g., $S1$. As such, an intent might be something like "On my network, I want to prioritize the traffic of Server $S1$". Our NLP preprocesses this sentence containing operator's intent tokenizing all words and performing the Porter' stemming algorithm. So the output from this initial preprocessing phase would be "network, priorit, traffic, server, s1". After that, we converted this tokens into numbers so that we could apply our ML algorithm. This task is performed using an encoded vector with dictionary of words.

In our case, our tokens would be:

```
{'network': 1, 'priorit': 2,
'traffic': 3, 'server': 4,
's1': 5}.
```

After this conversion, our MLP model predicts the output value for the encoded tokens, returning an array of values where each cell corresponds to the servers of the network. Still assuming the network consists of 4 servers, our output would be "[2 2 1 1 1]", where the first value corresponds to the main goal of the intent taken from the dictionary, and the other values correspond to the priority of servers' traffic.

After this phase, our Intent-Driven API processes this vector and append the traffic priorities in the configuration files that are subsequently used and manipulated by P4 during its initialization process. At run-time, the switch uses these weights for forwarding packets directed to/from $S1$ with higher priority.

V. Evaluation

This section describes our experimental settings and results obtained over a virtual testbed, with particular focus on how NLP4 works in the use case of intent-based load profiling.

A. Evaluation settings

Specifically, to test NLP4, we used Mininet, a network emulator that gives the possibility to create virtual networks and use them as a testbed for simulation purposes. As known in [31], Mininet is specially adapted to deploy software-defined networking (SDN) solutions, also using P4 as a language to simulate various topologies and case scenarios. We have set up a datacenter-like topology network with 10 servers connected to their switches which are consequently connected to other 4 switches, in a leaf-spine fashion, as shown in Figure 2. We set the links in the topology to have 100 Mbps bandwidth.

For the language interpretation process, we considered a MultiLayer Perceptron (MLP) algorithm with two connected hidden layers. The first hidden layer consists of 20 neurons and uses the rectified linear unit activation function; the second and last layer uses the sigmoid activation function. In our MLP model, the number of neurons is defined by the number of elements to be profiled: they can be the servers of the network, the switches, or the ports of a specified switch. In the following, we consider the profiling of servers, and, since our topology consists of 10 servers, the last layer of MLP has 10 neurons. Our model's number of layers, i.e., 2, and neurons are set after testing the accuracy metric in a preliminary performances analysis.

B. Evaluation results

In the following, we test the applicability of NLP4 when it has different profiles of traffic to maintain in the network. After converting load profiling intents into weights reported in the configuration file, our P4 switches perform a weighted choice when selecting the output links, so to match the traffic profile chosen by the user. Load balancing can be one particular case of traffic profiling, in which the network is optimized so that the load is equally distributed among the various links attached to the switch. Otherwise, in the more general case, the switch directs the outgoing traffic according to the weights assigned to its ports: ports with a higher weight are chosen more than ports with a lower one.

To test if the network's nodes can maintain the desired profile, we generate traffic (ping requests) sending 1000 packets between two servers not in the same leaf. In Figure 3, we compare the desired profile, specified by the user, and the actual profile that we obtain from the experiments. Such desired link load, only related to the leaf switches, are designed for testing purposes and are randomly chosen. From the figure, it is visible that the weighted port selection is coherent with the expected profile through each port, and the desired traffic is well maintained.

We then measure the overhead introduced by the execution of NLP4, and we compute the execution time of the program at varying the number of switches in the topology (Figure 4). We can observe how as the number of switches in the network increases, NLP4 maintains a limited execution time. This proves that even in a real data-center scenario, where the number of switches may be considerable, the execution time keeps being in the order of milliseconds. However, it must be noted that our program should run only when we want to change the load profile, whose occurrence is not very frequent.

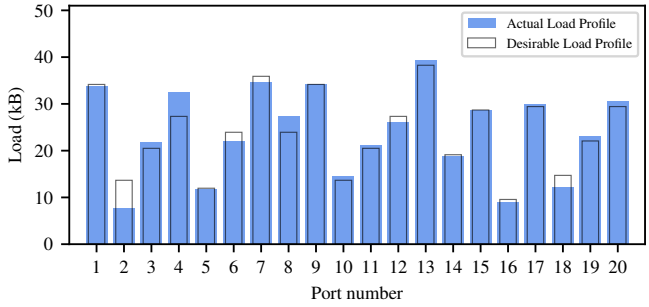

Fig. 3: Traffic is profiled across ports following the declared intent translated in P4 programs.

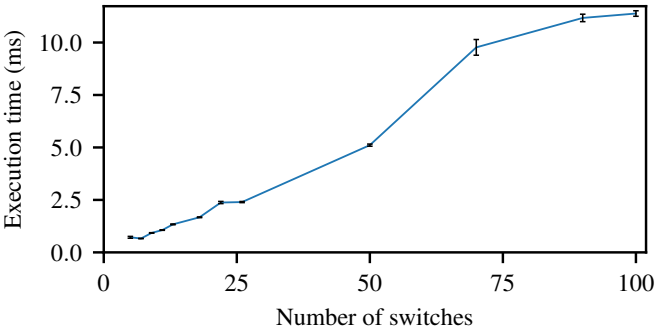

Fig. 4: Intent parser's overhead (execution time) as the number of P4 switches grows.

VI. Conclusion

In this paper, we presented NLP4, an intent-based solution that makes P4 data-plane programmability easier to implement thanks to a combination of Natural Language Processing (NLP) and MultiLayer Perceptron (MLP) techniques which led to the definition of an Intent-Driven solution. Through NLP4, even an inexperienced user can have his own customized network by making programmable switches behave as specified by the intent. Our preliminary results validate the validity of our approach in the case of load profiling specifications. Future work will consider more use cases, a more exhaustive dataset of possible intents for the user, and generalization of this solution to allow more switches actions.

References

[1] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela, "A survey of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 2, pp. 7–23, 1999.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[3] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, 2009, pp. 1–10.

[4] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," *ACM Sigplan Notices*, vol. 46, no. 9, pp. 279–291, 2011.

[5] G. Barel and R. Herwig, "Netcore: a network propagation approach using node coreness," *Nucleic acids research*, vol. 48, no. 17, pp. e98–e98, 2020.

[6] M. Hogan, S. Landau-Feibish, M. Tahmasbi Arashloo, J. Rexford, D. Walker, and R. Harrison, "Elastic switch programming with p4all," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, 2020, pp. 168–174.

[7] S. Ghorbani and B. Godfrey, "Towards correct network virtualization," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, 2014.

[8] R. Beckett, X. K. Zou, S. Zhang, S. Malik, J. Rexford, and D. Walker, "An assertion language for debugging sdn applications," in *Proceedings of the third workshop on hot topics in software defined networking*, 2014, pp. 91–96.

[9] J. Rischke and H. Salah, "Chapter 6 - software-defined networks," in *Computing in Communication Networks*, F. H. Fitzek, F. Granelli, and P. Seeling, Eds. Academic Press, 2020, pp. 107–118. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128204887000189

[10] M. Hogan, S. Landau-Feibish, M. T. Arashloo, J. Rexford, and D. Walker, "Modular switch programming under resource constraints," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/hogan

[11] E. Zeydan and Y. Turk, "Recent advances in intent-based networking: A survey," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. IEEE, 2020, pp. 1–5.

[12] M. Riftadi and F. Kuipers, "P4i/o: Intent-based networking with p4," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 438–443.

[13] I. Matta and A. Bestavros, "A load profiling approach to routing guaranteed bandwidth flows," in *Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No. 98*, vol. 3. IEEE, 1998, pp. 1014–1021.

[14] F. Esposito, J. Wang, C. Contoli, G. Davoli, W. Cerroni, and F. Callegati, "A behavior-driven approach to intent specification for software-defined infrastructure management," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2018, pp. 1–6.

[15] Y. Li, C. Jia, X. Hu, and J. Li, "Mahjong: A generic framework for network data plane verification," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, 2021, pp. 52–58.

[16] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software defined networks," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 1–13.

[17] R. Soulé, S. Basu, R. Kleinberg, E. G. Sirer, and N. Foster, "Managing the network with merlin," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, 2013, pp. 1–7.

[18] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville, "Refining network intents for self-driving networks," in *Proceedings of the Afternoon Workshop on Self-Driving Networks*, 2018, pp. 15–21.

[19] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 85–98.

[20] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "Pga: Using graphs to express and automatically reconcile network policies," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 29–42, 2015.

[21] A. Alsudais and E. Keller, "Hey network, can you understand me?" in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2017, pp. 193–198.

[22] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev, "{Net2Text}:{Query-Guided} summarization of network forwarding behaviors," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 609–623.

[23] A. Gatt and E. Krahmer, "Survey of the state of the art in natural language generation: Core tasks, applications and evaluation," *Journal of Artificial Intelligence Research*, vol. 61, pp. 65–170, 2018.

[24] V. Balakrishnan and E. Lloyd-Yemoh, "Stemming and lemmatization: a comparison of retrieval performances," 2014.

[25] I. Boban, A. Doko, and S. Gotovac, "Sentence retrieval using stemming and lemmatization with different length of the queries," *Advances in Science, Technology and Engineering Systems Journal*, vol. 5, no. 3, pp. 349–354, 2020.

[26] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[27] H. Ramchoun, Y. Ghanou, M. Ettaouil, and M. A. Janati Idrissi, "Multilayer perceptron: Architecture optimization and training," 2016.

[28] L. P. Binamungu, S. M. Embury, and N. Konstantinou, "Maintaining behaviour driven development specifications: Challenges and opportunities," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 175–184.

[29] T. Storer and R. Bob, "Behave nicely! automatic generation of code for behaviour driven development test suites," in *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2019, pp. 228–237.

[30] A. V. Ventrella, F. Esposito, and L. A. Grieco, "Load profiling and migration for effective cyber foraging in disaster scenarios with formica," in *4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 80–87.

[31] B. Lantz and B. O'Connor, "A mininet-based virtual testbed for distributed sdn development," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 365–366, 2015.