When NLP meets SDN : an application to Global Internet eXchange Network

Manh-Tien-Anh Nguyen*‡, Sondes Bannour Souihi‡§, Hai-Anh Tran*, and Sami Souihi†
*School of Information and Communication Technology
Hanoi University of Science and Technology, Hanoi, Vietnam
†LISSI-TincNET Research Team University Paris-Est Creteil, France
§ CEA LIST, University of Paris-Saclay
‡TexPECT, France
Email: anh.nmt187213@sis.hust.edu.vn, sondes.souihi@cea.fr, nhth@soict.hust.edu.vn, sami.souihi@u-pec.fr

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9838633

***Abstract*—Software-Defined Networking (SDN) and its extension Intent-Based Networking (IBN) are network paradigms that enable dynamic, programmatically efficient network configuration. IBN allows network operators to express an outcome or business objective without the low-level configurations necessary to program the network to achieve these demands. Existing research proposals for IBN introduce several systems to translate users intents into network infrastructure configurations. Despite the positive aspects of these proposals, they still suffer from many drawbacks. Some require users to learn a new intent definition language. Some others may lack the appropriate grammar to make these frameworks recognize the intent correctly. In this paper, we introduce a framework leveraging the capabilities of Natural Language Processing (NLP) for network management from an operator utterances. In order to understand natural language, our framework uses the sequence-to-sequence (seq2seq) learning model based on recurrent neural networks (LSTM). The model has been improved by using word embedding and user feedback. As a proof of concept, we implement our framework for network management in a Global Internet eXchange Network and evaluate its practicality regarding NLP accuracy and network performance.**

***Index Terms*—Software-Defined Networking (SDN), Intent-Based Networking (IBN), Natural Language Processing (NLP), Artificial Intelligence (AI), Internet eXchange Points (IXP)**

I. Introduction

In recent years, Software Defined Networking (SDN) [1] has gotten much interest from academics and industry to design network management systems. SDN technology is a network management strategy that allows for dynamic, programmatically efficient network configuration. It decouples the control plane from the data plane and centrally integrates the network logic at the controller level. This programmable architecture makes network innovation and development much more straightforward. However, the advantages offered by SDN may reach their limits in the context of very large-scale networks.

Driven by technology and demand, Intent-Based Networking (IBN) was proposed to overcome the shortcoming of SDN architecture. An IBN is an intelligent network that can automatically convert, verify, deploy, configure, and optimize itself to reach the desired network state based on the operators requirements and automatically resolve abnormal events to maintain network stability [2]. Furthermore, IBN allows network administrators to express an outcome or business objective – the intent – that dictates how the network should behave without the low-level configurations necessary to program the network to achieve these demands. IBN architecture relies on Natural Language Processing (NLP) to create an abstract, cognitive layer to bridge the gap between network management and the operator's intent. Existing research proposals for IBN introduce several structured network intent languages, frameworks, and systems to translate users intents into network infrastructure configurations [4]–[7]. While these proposals are positive milestones, some intent languages require users to learn a new intent definition language in each proposal, and the others may lack the grammar to make these frameworks recognize the intents correctly. Moreover, most proposals for an abstract layer do not support the feedback feature, which requires the user to confirm the output intent generated by their system and increase the predicted intent accuracy after each user's requirement.

In this paper, we present a framework that leverages the capabilities of Natural Language Processing (NLP) to manage and configure Networks from operator's utterances. Our framework includes four main stages. Firstly, we create a chatbot to classify the intent and recognize the named entities from the operator's utterance expressed by Natural Language. We rely on Dialogflow [3], which uses word embedding for the intent classification stage, and machine learning for the named entities recognition stage to develop our chatbot interface. Secondly, we apply a sequence-to-sequence (seq2seq) learning model with neural networks [9] to convert the recognized entities into a high-level structured network language called SNIL (Section IV). Next, the structured language is then sent to the user for feedback on the recognized entities. Finally, we easily convert the structured network language into a network policy for managing and configuring in Global Internet eXchange Network (GXN) [8] that we considered a use case. A GXN is a network of interconnected Internet eXchange Points (IXP). For implementation, we use a software-based controller Open Network Operating System (ONOS), as its controller. Our system relies on the REST-API interface to communicate with the ONOS controller and convert the structured network language into ONOS intent.

The remainder of this paper is structured as follows. In Section II, we review related work. In Section III, we present our framework and its design using natural language processing and machine learning for network management from operator's utterances. We propose a structured network intent language which is similar to natural language to serve as an abstract network layer in Section IV. In Section V, we detail the evaluation of our framework, and we conclude in Section VI.

II. Related Work

In this section, we review related work. We consider as related work previous efforts which combine SDN and intent.

In fact, given the lack of strategies in this area, we review intent languages that allow deploying network configurations using high-level directives on SDN.

Major SDN controllers, such as the Open Network Operating System (ONOS) and OpenDayLight (ODL), implement IBN capabilities that abstract the details of specific network protocols, mechanisms, and topology. Intents allow network applications or practitioners to specify declaratively what the network ought to do at a policy level and how the network ought to achieve the desired configuration. For example, ONOS has an Intent Framework component [2], while a network abstractions module of OpenDayLight is called Intent Component [2].

Kiran et al. [4] developed the Intelligent Network Deployment Intent Renderer Application (iNDIRA) system to enable intent-based networking. iNDIRA uses natural NLP to allow users to express their intents, converted into Resource Description Framework (RDF) graphs before getting translated into low-level commands. Alsudais et al. [5] proposed an intermediate network-agnostic layer that acts as the medium between natural language input (spoken or written) and different network implementations. They have leveraged the programmability that SDN offers to build a prototype tool that takes natural language text as an input and builds abstract tasks. Such tasks are then passed to a network controller to be performed in real-time. Esposito et al. [6] presented a northbound interface solution for network intent specification based on Behavior-Driven Development. Their approach allows intent expressiveness in English, Mandarin, or any other natural language, by leveraging the Gherkin programming language. Their prototype includes an intent processor, an intent-policy interpreter, and several Software-defined infrastructure-specific policy actuators. Jacobs et al. [7] introduced a novel intent refinement process and Nile, a high-level intent definition language, aiming to be a step towards enabling self-driving networks. The proposed refinement process leverages a user-friendly chat interface and a model that extracts from natural language to a structured intent program, written in Nile. The extracted Nile intent acts as an abstraction layer for lower-level configuration and policy languages, which allows them to ask for feedback from the operator before compiling the structured intent into network configurations.

While these approaches gain a lot of significant achievements, some intent languages require users to learn a new intent definition language, and others may lack the grammar necessary for these frameworks to recognize the intents correctly. Furthermore, the feedback feature, which requires the user to check the output intent created by their framework and raise the predicted intent accuracy after each user's demand, is not supported by most abstract layer ideas. In addition, these proposals can not clearly distinguish between the intent for retrieving the network information and the intent for submitting the processed data to a destination network and not being implemented in large-scale networks.

III. Framework Overview

Figure 1 presents an overview of our process with four main stages: intent classification & named entities recognition, intent conversion, intent confirmation, and intent implementation. At the first stage, we develop a chatbot to classify the intent and recognize the entities from the operator's utterance expressed by Natural Language. We rely on Dialogflow [3] which uses word embedding for intent classification and machine learning for entities recognition to implement the chatbot interface. In our chatbot, we divide the user's intent into GET intent and POST intent. These two intent have the same entities' properties: username, network endpoints, actions, QoS, and temporal configurations for the policy.

Secondly, we use a sequence-to-sequence learning model with neural networks [9] to convert the recognized entities from the previous stage into a high-level structured network language called SNIL (Section IV).

Thirdly, the structured language is then sent to the user for confirmation on the recognized entities with its intent type. The operator may check the accuracy of the output intent generated by our system and make adjustments if essential. These stages are repeated until the operator confirms the correct translation of the intents.

Finally, we compile the structured network language confirmed by the user into a network policy for managing and configuring a Global Internet eXchange Network. For the deployment, we use a software-based controller: Open Network Operating System (ONOS). We use the REST-API interface to communicate with the ONOS software-based controller. Our framework retrieves the required information, converting Natural Language intents into ONOS intent, and transmits it through this interface.

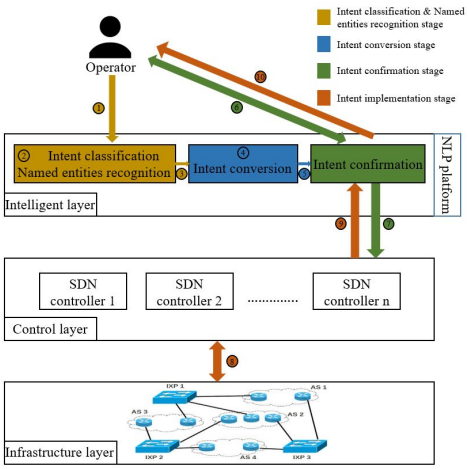

Fig. 1. Framework Architecture.

A. Intent classification & Named entities recognition (IC & NER)

First of all, we need to classify the intent and recognize the actions with important information of the network activities from the operator's utterances. Moreover, we have to construct a chatbot interface to interact with users, get users' intents, and respond to the users. In this step, we use a chatbot development platform to build the IC & NER. Many state-of-the-art chatbot platforms exist, such as Amazon's Lex, Google's Dialogflow, IBM Watson Assistant, and Azure Bot Service, but we decide to rely on Dialogflow to construct our chatbot because of its convenience and reusability. Dialogflow, previously known as API.ai, is a Google-owned developer of human-computer interaction technologies based on natural language conversations. Machine learning is used in the framework to generalize sample instances known as *entities* and to make feature recognition easier in the dialog. In our

chatbot, the *entities* are username, actions, network endpoints, SLA requirements, and temporal constraints. Figure 2 presents the attributes and its example of an *entity*. Moreover, the framework uses word embedding to classify intent. To further assist us in determining the type of intent, we take the idea of Representational State Transfer (REST) from web services implementation. We divide the operator's utterances into GET intents and POST intents. The POST intents are used to distinguish actions that add some network configuration or require SLA into the network. The GET intents are used to retrieve network information.

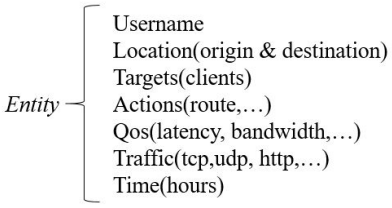

Fig. 2. Entity Definition.

In natural language processing (NLP), word embedding is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning. Word embeddings can be obtained using a set of language modeling and feature learning techniques where words or phrases from the vocabulary are mapped to vectors of real numbers. Conceptually it involves the mathematical embedding from space with many dimensions per word to a continuous vector space with a much lower dimension.

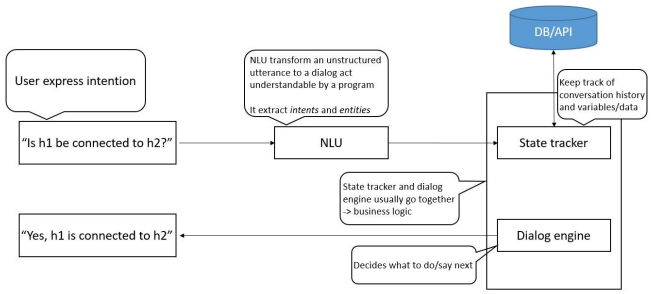

Fig. 3. Intent Classification Workflow.

In the Dialogflow platform, an intent classifier is made up of a few elementary components: some preprocessing pipeline (mostly made up of a tokenizer), an embedding vectorizer, and a simple classifier. Figure 3 illustrates the workflow of intent classification. All of the magic here lies in the embedding vectorizer. The embedding vectorizer makes use of large pre-trained GloVe or FastText vectors [9]. These vectors have been trained on large corpora made up of millions of documents. This means that these word vectors are very well tuned to have words with similar meanings very close to each other in vector space. When we train a classifier that makes use of these embeddings, most of its work is already cut out for it. Since similar words will have very close word vectors, the classifier can generalize more easily and assign the same label to words with close vectors. Figure 4 presents how Word Embedding is used in intent classification.

When we build an NLP model from scratch, we initialize our embeddings randomly. Then we train it to distinguish words with different meanings by grouping them. This requires training the model on millions of sentences. For this reason, we rely on Dialogflow for the first step, as it allows us to refine a pre-trained model with minimal data.

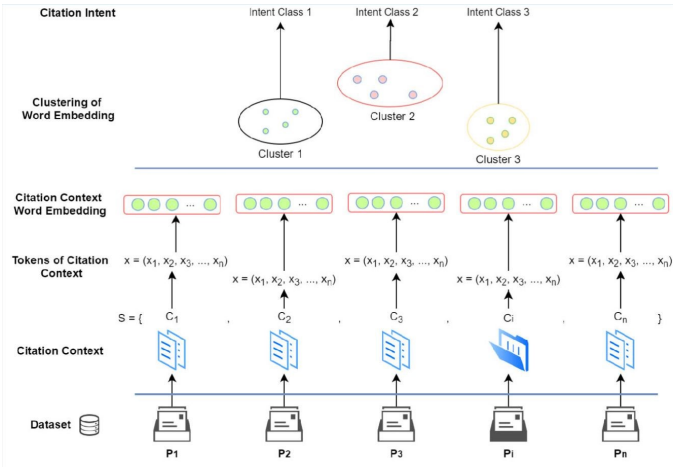

Fig. 4. Intent Classification using Word Embedding.

Even while chatbots are highly beneficial for user interactions, they do not meet all the intent-based network planning requirements. The user utterances are represented by key-value pairs derived from the entities retrieved from natural languages. These pairings, on the other hand, do not recognize the network configuration commands. For example, if a network operator asks a chatbot to "make a route from h1 to h2" the chatbot will respond, depending on how it is built. A possible extraction result might be the following entities: origin: 'h1', destination: 'h2', action: 'route'. As a result, we still need to convert the entities into a structured intent that can be executed in a destination network following the chatbot interaction.

B. Intent Conversion

After the chatbot interface classifies the intent and collects all the necessary *entities* from the operator utterances, we need to send this information to another module in real-time to perform the second step. Thus, we rely on WebHook, which is based on REST-API, to send these *entities*. A webhook (also called a web callback or HTTP push API) is a way for an app to provide other applications with real-time information. A WebHook delivers data to other applications as it happens, meaning we will get data immediately. To get all of the recognized *entities*, we set up a WebHook from our chatbot to our Intent Conversion. These *entities* are put into a previously trained sequence-to-sequence learning model, which converts them into structured intent language. We rely on NILE language [7] to represent structured language called SNIL with some extensions and modifications of its grammar (detailed in Section IV). We adjust it in order to make SNIL have ability to distinguish between the intent for retrieving the network information and the intent for submitting the processed data to a destination network, as well as can perform multiple actions that common in network management.

As mentioned above, our framework uses the sequence-to-sequence model to convert the recognized entities to structured intent language similar to NILE language. Sequence-to-sequence models are deep learning models that aim to map a fixed-length input with a fixed-length output where the input and output length may differ. The model consists of an encoder, an intermediate (encoder) vector, and a decoder. The encoder uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence (in our case, the recognized entities) to a vector of a fixed dimensionality, which aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions. The decoder uses another deep LSTM to decode the vector's target sequence (structured intent language). The encoding-decoding process is illustrated in Figure 5. One of the essential advantages of RNN architecture is that it allows different input lengths and output sequences.

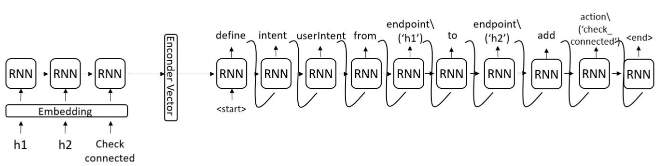

Fig. 5. Seq2seq learning model.

One of the drawbacks of utilizing neural networks for text-to-text translations is the vast vocabulary of each language, which necessitates big datasets and a significant amount of time for the models to be trained. We can solve the above limitation by anonymizing the entities and using previously recognized entities as input and a restricted, well-defined language as output. This pre-processing replaces each recognized entity with a token representation and feeds the token representation to the RNN encoder as input. For instance, if the outputs of recognized entities are "route" and "host 1", we would utilize anonymization to convert them to the tokens' @action' and '@location' before starting the Intent Conversion stage. Then, we deanonymize the resultant intents to replace the tokens with the initially recognized entities. Thus, we may drastically minimize the number of training instances required for the model by utilizing anonymization because we don't have to examine every potential entity value for network intents.

After performing entities' anonymization, we transform each input word to a unique numerical representation since we cannot utilize words directly as input for the sequence-to-sequence model. The numeric indices in a pre-built dictionary containing all words in the model represent the anonymized entities numerically. Figure 6 shows a list of entities' anonymization and its corresponding numerical representation.

[@username, @action , @target, @location, @traffic, @qos_metric, @qos_constraint, @qos_value, @hour]
[1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9]

Fig. 6. Entities' Anonymization and its numerical representation.

In addition, we implement Word Embedding vectorization in the first layer of the RNN encoder to improve the learning rates and prediction accuracy of our models. Word Embedding vectorization will create an array of real values from a sequence of anonymized entities, which is then parsed into the encoder layer to generate the encoder vector. The RNN decoder then uses the encoder vector as an input and translates it into SNIL language. The structured language created by the decoder layer is then sent to the Intent confirmation module for the next stage.

C. Intent Confirmation

After receiving the structured network language from Intent conversion, Intent confirmation will send this result to the network operator via the chatbot interface for confirmation of the recognized desired intent. The operator may check the accuracy of the output intent generated by our system and make adjustments if essential. The operator can edit the intent until the structured network language satisfies the requirement. After the operator's feedback, the corrected intent referred from our system is then sent to the target Controller for implement it into the network infrastructure. Moreover, the corrected intents are involved in the training dataset of the seq2seq model, and a new training process is started. This process gives our system better accuracy and guarantees that the outcomes improve every time the user requires an intent because we continuously update the weights for our models. The new output intent referred from the following user utterances will be predicted using the latest weights we received.

D. Intent Implementation

Let us suppose that an operator wants to establish a route from one point to another with a specific service level agreement (SLA) (an amount of bandwidth, traffic, latency,...). For this, the operator needs to discover all network devices, gather all the information related to SLA, and understand their respective configurations. However, it is not easy to collect all the necessary information in a large-scale network.

Instead, the operator has to express their intent using natural language to our framework. Then it automatically translates the operator's utterances into network configurations. This section analyzes a case study to make a route from one point to another with specific SLA in our network topology. We use Mininet simulator to create a GXN composed of multiple interconnected IXPs.

For instance, if a network operator asks a chatbot, "Establish a route from h1 to h2, with latency less than 10ms and bandwidth more than 400Mbps, and allow UDP only", our framework will extract the entities from this intent then translate it into SNIL intent. Hence, it will send the feedback to the operator and repeat it until the operator confirms the correct translation of the intents. Finally, it will convert SNIL intent into ONOS intent through REST-API and deploy it to the destination network.

IV. SNIL: Structured Network Intent Language

At the previous stage, we have explained the process of implementing user requirements expressed by natural language into network infrastructure and then returning the result to the operator. However, we did not mention one of the most important elements from this process, the abstract network layer. While network management requires operators to have experience and expert skills in the network domain, natural language is difficult to parse and implement directly into a network infrastructure. Therefore, there is a missing link in the interaction between network management and natural language. Fortunately, the abstract network layer bridges the gap

between network management and natural language expressed by operators and home users. The related work section has discussed many proposed languages for the abstract layer with many advantages for each proposed language. With all the state-of-the-art proposed languages that we have explored, we want to create a new structured network intent language with as many advantages as possible. We propose SNIL (Structured Network Intent Language) which is similar to NILE language [7] to serve as abstract network layer language. Here, we modify and extend this original version to satisfy network management requirements.

Table I presents the main operation from our structured network intent language. Two operations are compulsory: from/to and add operations. All of the network intents must have two operations to meet the complete requirements. The other operations are used when we want to request the intent with POST type, and therefore these operations are optional. To guarantee the intent that satisfies syntax and semantics of the structured network language with full information, we rely on NILE [7] grammar and required operations for each intent that we receive. If the operator expresses the intent with wrong grammar or lack of information, the system warns the operator via the chatbot interface to correct the intent. For instance, if the operator expresses the intent: "Make a route", the system will require the operator to provide from/to operation to obtain the correct intent. Although this intent satisfies the semantic of the language, it does not have all the required operations. The confirmation stage will finish until the operator corrects the intent.

Operation	Attribute	REST API Type	Required
from/to	location	GET/POST	Yes
add	action	GET/POST	Yes
put	bandwidth/latency	POST	No
allow/block	traffic/service/protocol	POST	No
start/end	hour/date/datetime	POST	No

TABLE I
OVERVIEW OF STRUCTURED NETWORK INTENT LANGUAGE OPERATIONS.

With SNIL, the operator can express complex intents intuitively. For example, if a network operator asks a chatbot, "Establish a route from h1 to h2, with latency less than 10ms and bandwidth more than 400Mbps, and allow UDP only", our framework will recognize the entities from this intent and translate it into structured intent language.Note that mapping endpoints between logical and physical names (e.g., h1 and h2) must be resolved during the implementation stage, as they represent information specific to each network. This allows users to express the intent in an abstract style (not to specify IP addresses, but instead say something abstract like "h1"). This example illustrates how expressive grammar the structured intent language has; therefore, we can rely on this structured language to represent many real-world network requirements.

V. EXPERIMENTS

In this section, we discuss our experimental results for both the NLP domain and network domain. All our experiments were done on machines with 6 core, 6 thread Intel(R) Core(TM) i5-9400F CPU at 2.90GHz, 16GB of RAM, AMD Radeon(TM) RX 570 8GB graphics card, running Ubuntu 20.04 LTS. The testing scenario is based on the ONOS controller version 2.6.0 and the Mininet version 2.3.0 to emulate the Global Internet eXchange Network on the full-meshed topology of 5 edge switches connected with one core switch for each IXP, totaling 3 IXPs interconnected. We relied on Quagga Software Routing Suite to simulate the BGP process at the participant's routers. Finally, we made communication between our system and ONOS controller through REST API that we had already set up.

A. NLP evaluation

To evaluate the NLP domain, we measured the system's accuracy that depends on training dataset size to discover the optimal result between dataset size and prediction accuracy. Because dataset size impacts the training time crucially, we determined the most optimal training dataset sizes for the highest prediction accuracy to reduce the training time. We automatically created the datasets with random sets of intent types with recognized entities and structured network intent language pairs. Each pair consists of a various number of actions, endpoints, SLA rules, and temporal constraints. All training iterations were implemented with 50 epochs, batch size of 64, and a validation split of 20%. We measured seven different sizes of training datasets: 200, 500, 1000, 3000, 5000, 7000, and 8000 pairs. We created a separate testing dataset for each training dataset size, which contains 20% of the training dataset's pairs. To evaluate the accuracy for each dataset size, we measured for each prediction in the testing dataset the coefficient of determination (R-squared) between the structured network language predicted by the model and the expected output structured network language. In our experiments, the more accurate the model is when the R-squared value is closer to 1, this means that the model predicts the output with high accuracy and fewer wrong results. The measurements created a list of R-squared values for each dataset size. Figure 7a presents the mean with 95% confidence interval of R-squared values for each size of training datasets. We can see that we can have more accurate results when we increase the size of datasets, but we will have a longer training time with larger datasets. From Figure 7a, it can be observed that we need to train the model with only 7000 pairs to achieve desired results in our model. We have not expanded the datasets to more than 8000 samples because, with only 7000 entries, we have reached excellent results from our model. Figure 7b shows the training times for each size of the dataset. Thus, to get the desired results, we have to train our model for approximately four hours for 7000 pairs, and the larger size of datasets require a longer training time.

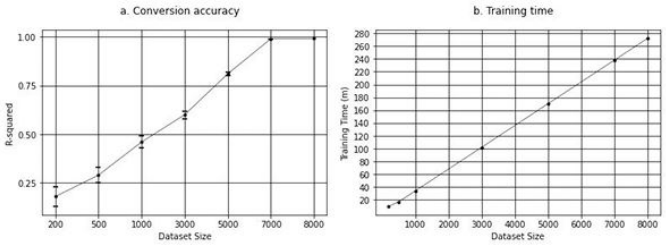

Fig. 7. Conversion accuracy and training time by each dataset sizes.

In addition, we measured the average response time of the system that depends on multiple attributes such as the intent type, the entity type, and the number of entities. The average response time that we have measured consists of compilation

time and deployment time. We divided the operator's intent into five groups and estimated the average response time for each group: GET intents, POST intents, only route action, route action with SLA requirements, and route action with traffic constraints. These are the most popular intent appearing in network management. There are two intent types in our system: GET intent and POST intent. These two intents have different response times due to their attributes. Most GET intents have response time lower than POST intents because POST intents have to process and be configured into the destination network, while GET intents have to gather information from the network before sending it to the operator. The other attribute that affects response time is the number of entities. Intent having many QoS requirements and traffic constraints will cause high response time than only routing action. We produced a dataset with 50 intents per group, totaling 250 different intents, and measured the mean response time of each group. This experiment was deployed on the Global Internet eXchange Network with about 400 network elements approximately. We relied on the Mininet emulator to create the network topology that interacted with the ONOS controller. Figure 8 presents the result that we received.

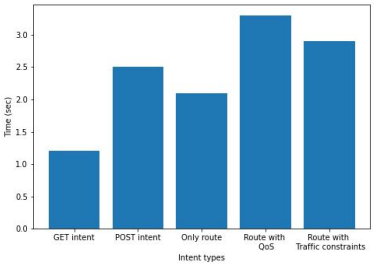

Fig. 8. Average response time of each intent.

B. Network evaluation

Next, we explain the evaluation results from the network domain of our system. To assess the feasibility of our system in the network domain, we compare the throughput improvement over time between using our system and manual configuration. First, we make a route from one host to another and measure throughput between two hosts over time. Next, we put some noise into this route and track the fluctuation of throughput. To evaluate the rapid adaptation when using the system, we require our system to improve the throughput through the chatbot interface immediately. Figure 9 compares the throughput improvement over time when using the system and configuring manually. The noise makes throughput between two hosts reduce significantly, from almost 30 Mbps per second to 15 Mbps per second. As expected, when using the system, it would discover the other optimal path with the highest throughput to make a route between two hosts, and it just takes approximately three seconds to improve the throughput to 30 Mbps per second again.

On the other hand, the dashed line in Figure 9 presents the throughput improvement over time when configuring manually. We had to find all paths that could make the route between two hosts manually for this stage. After that, we would choose the path that has the highest throughput and configure it. Because of these time-consuming processes, manual configuration takes about six-second to take another route that has the same throughput as the beginning. From Figure 9, we can see that throughput over time was dramatically improved while the other was changed marginally when using the system.

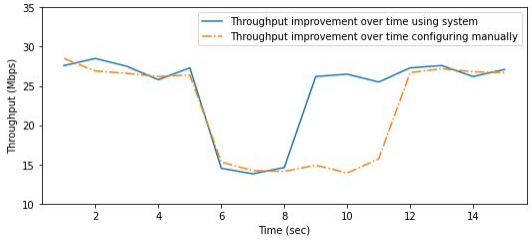

Fig. 9. Throughput improvement over time.

VI. Conclusion

In this paper, we proposed a framework that helps operators to express their intent with ease in Global Internet eXchange Network. The framework leverages a chatbot interface powered by Dialogflow and a sequence-to-sequence learning model that converts entities from natural language to a structured network intent language called SNIL. The structured intents are then mapped into network policies and processed to the destination network. We argued that by utilizing the capabilities of Natural Language Processing into a large-scale network such as the Global Internet eXchange Network, network administration should be more convenient and interesting. Our experiments on the NLP domain witnessed the average response time of the system with high-speed for each type of intents and the effect of dataset size on the translation accuracy. In addition, we evaluate the throughput improvement over time between our system and manual configuration to show the effectiveness of our framework in the network domain.

Acknowledgment

This research is partially funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.02-2019.314

References

[1] Bannour, F., Souihi, S., and Mellouk, A. (2017). Distributed SDN control: Survey, taxonomy, and challenges. IEEE Communications Surveys and Tutorials, 20(1), 333-354.
[2] Pang, L., Yang, C., Chen, D., Song, Y., and Guizani, M. (2020). A survey on intent-driven networks. IEEE Access, 8, 22862-22873.
[3] Google Inc. Dialoglow. [Online]. Accessed on: Oct. 21, 2021. Available: https://dialoglow.com/
[4] Kiran, M., Pouyoul, E., Mercian, A., Tierney, B., Guok, C., and Monga, I. (2018). Enabling intent to configure scientific networks for high performance demands. Future Generation Computer Systems, 79, 205-214.
[5] Alsudais, A., and Keller, E. (2017, May). Hey network, can you understand me?. In 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (pp. 193-198). IEEE.
[6] Esposito, F., Wang, J., Contoli, C., Davoli, G., Cerroni, W., and Callegati, F. (2018, November). 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) (pp. 1-6). IEEE.
[7] Selle Jacobs, A., José Pfitscher, R., Alves Ferreira, R., and Zambenedetti Granville, L. (2020). Refining Network Intents for Self-Driving Networks. arXiv e-prints, arXiv-2008.
[8] Scholte, J. A. (2018). Internet governance. In The Routledge Handbook of Transregional Studies (pp. 311-319). Routledge.
[9] Tripathy, J. K., Sethuraman, S. C., Cruz, M. V., Namburu, A., Mangalraj, P., and Vijayakumar, V. (2021). Comprehensive analysis of embeddings and pre-training in NLP. Computer Science Review, 42, 100433.