To All Intents and Purposes: Towards Flexible Intent Expression

Mehdi Bezahaf, Eleanor Davies, Charalampos Rotsos, Nicholas Race
Schoole of Computing and Communications
Lancaster University
firstname.lastname@lancaster.ac.uk

***Abstract*—Intent-based networking provides an efficient mechanism to manage complexity in network management. The paradigm allows users to express their network requirements, and an autonomic framework translates them into a network configuration. Existing efforts focus primarily on modeling connectivity intents for end-users. Nonetheless, in order to deliver autonomic behavior in network management, an intent system must support a wider range of network management processes and model human-to-human interactions, essential for network operation. Furthermore, such interactions may involve non-technical users and require the design of novel interfaces, supporting free-text and conversational intent expression. Towards this goal, we present an intent architecture that supports novel network management intents, such as network path rerouting and applying periods of 'service protection'. The paper includes details of our prototype implementation that is capable of deploying such intents in under five seconds in a large mininet topology.**

I. Introduction

Network management complexity in recent years has increased as a result of the exponential growth of global Internet traffic and the size of network infrastructures. New networking paradigms, such as Software Defined Networking (SDN) and Network Function Virtualization (NFV), offer potential improvements in the ability of network operators to manage infrastructures, through the use of network programmability [1]. Alongside the promise offered via the introduction of programmability, it is also essential to develop flexible policy expression mechanisms, capable of capturing user network requirements and mapping them into a network configuration.

Intent-based networking (IBN) [2] is a novel network paradigm that automates the process of network configuration. Users can describe their functional goals using a high-level API or language and the intent system will adapt the network configuration to deliver these goals. As a result, the operator management complexity is reduced and the abstraction level of objectives is increased. Autonomic behavior can be achieved by enabling intelligence, using machine learning (ML)/optimization algorithms, in the intent translation process, and compute an optimal network configuration, given the operation status of the infrastructure (*e.g.* topology, traffic matrix).

Currently, several intent-based network data models and optimization frameworks have been proposed in the literature [3], [4], [5], [6]. Relevant frameworks allow users to express their network intents using domain-specific languages (DSL) or APIs and translate them automatically into appropriate network configurations.

The adoption of a fixed syntax and vocabulary for the expression of network intents simplifies the translation task and enables several added-value properties for the intent framework, including verifiability, automatic adoption, and fast service delivery. Nonetheless, these benefits can also significantly hinder the usability of intents, particularly by users who have limited technical knowledge. In order to take full advantage of these benefits, users must become familiar with the syntax of the programming language or data models. Users must also develop familiarity with programming thinking and understand networking concepts to master the semantics of language primitives.

The importance of this challenge will increase especially as the application domain of network intents expands. For example, network configuration represents only a fraction of the coordination required to deliver a service by an ISP, which involves several human processes, including logistics and billing. Integration of such processes requires the expansion of existing intent languages with new semantics and the introduction of mechanisms to efficiently translate ambiguous user input into intents. Existing intent systems exclusively target connectivity intents.

In this paper, we argue that true autonomic operation for future networks must integrate human processes through the definition of new intent primitives, beyond simple matters of connectivity. To reduce the complexity of network intent expression, new user interfaces should be designed to reduce the cognitive load for non-technical users and efficiently handle ambiguity in user input. To support this objective, we present an intent system with novel intent primitives. The system exposes these intents via a conversational intent expression interface using off-the-shelf Artificial Intelligence (AI) and Natural Language Processing (NLP) tools and can interact with several online services, like Slack and Google Assistant. Finally, our architecture can automatically translates user intents into network configurations.

Specifically, this paper has the following contributions:

- We present an intent system architecture capable of translating user free-text input into appropriate network configurations.

978-1-6654-0522-5/21/$31.00 ©2021 IEEE

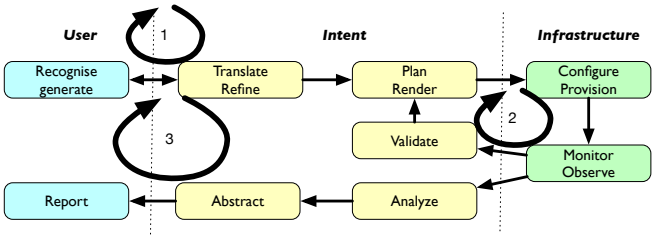


Fig. 1: Intent systems control loops.

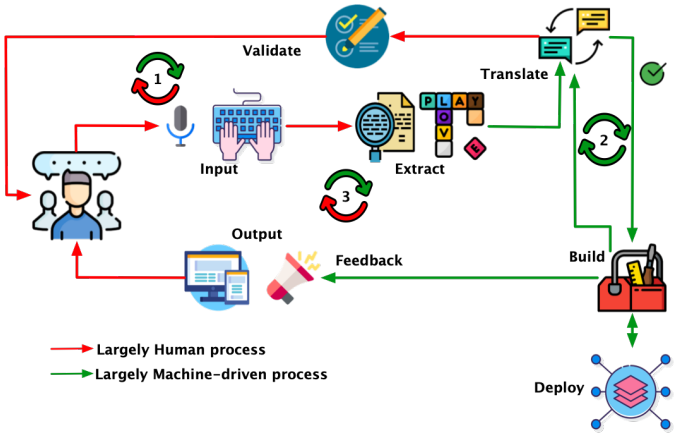


Fig. 2: Our system architecture employs a micro-service approach and splits the translation process into several stages. Intent deployment is driven by three basic control loops

- We demonstrate the ability of our system to capture novel intent use-cases (service protection, path rerouting) and to deploy them in less than 5 seconds.

For the remainder of this paper, we discuss related literature (§ II) and motivate our system (§ III). Furthermore, we present our architecture and describe our prototype implementation (§ IV). Finally, we demonstrate our system in operation and measure the incurred processing latency (§ V), before concluding our work (§ VI).

II. Related Work

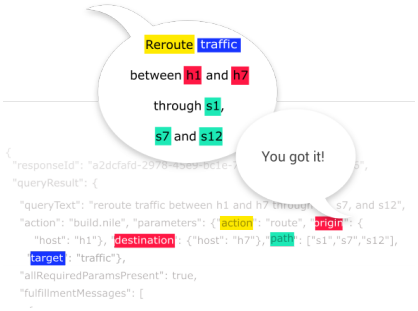
An intent system organizes intent processing using three planes: the *User*, the *Intent*, and the *Infrastructure*, depicted in Figure 1. These planes coordinate through three control loops, to ensure intent fulfillment and provide feedback to the user. [7] An initial control loop (loop 1) is established between the user and the intent layer, designed to capture and verify user intent details. The intent layer uses a separate control loop (loop 2) to adapt the network configuration, based on monitoring information from the infrastructure, and fulfill the intent goals. This control loop uses network telemetry information to measure the available network resources and validate the feasibility of each intent. This control loop can be static or dynamic, depending on whether the intent framework re-evaluates intent fulfillment post-deployment. Finally, a control loop (loop 3) exists between the user and the intent layer, which abstracts monitoring information and the intent state and provides useful feedback to the user.

Research efforts on IBN have primarily focused on the design of the second control loops from Figure 1. Merlin [4] is an early attempt to express network policies using a declarative language and map them into an SDN configuration. PGA [3] is a policy framework for OpenFlow networks, providing conflict-free intent composition using graph theory. Chopin [5] is an intent framework for SDN networks, which supports the expression of connectivity intents with QoS. Propane [8] is a declarative language, allowing the mapping of connectivity intents in legacy network control protocols, such as BGP. Robotron [9] is an intent-based management system used to manage the Facebook network infrastructure. At its core, Robotron uses a data model to track both policy specification, as well as, the device inventory of the underlying infrastructure. Network operators can use this model to automate the policy translation into appropriate device configuration, by developing vendor-specific drivers. Relevant efforts focus on the delivery of connectivity intent and lack any insights on how additional management processes can be automated via an intent framework.

In terms of the first and the third control loop, and the design of user interfaces for IBN, research has been somewhat limited. Alusdais and Keller [10] presented the first attempt to design a system to extract user intent from verbal input. The system uses NLP tools to parse and translate text into NBI calls to a network controller. NILE [11] is a user-friendly intent language, offering automatic deployment of connectivity intents with middlebox support. The platform is open-source and provides a conversational interface through the Google Assistant service. In parallel, the system uses a simple and extensible intent expression language, which is user-friendly. Bezahaf *et al.* [12] present an intent framework that self-adapts deployed intents and sends feedback to the user about new intent updates. Understanding the design of these interactions and developing mechanisms that put the user-in-the-loop, can open up new approaches in the design of IBN systems, which improve automation by asking user for precise feedback to reduce ambiguity and simplify intent delivery.

III. Capturing User Intent

Research on IBN systems has predominantly focused on supporting connectivity intents through autonomic network management systems. However automatic network configuration reflects only a fraction of the functionality of a modern ISP. Service delivery requires the coordination of several processes within an ISP. Such processes require human-to-human interaction to translate input from non-technical users into changes to device configuration or online workflow processes. The adoption of programming languages and network-specific data models to abstract network intents, makes these interaction 'interfaces' inaccessible for non-technical users



(a) Dialogflow processes syntax-free user input and extracts key terms.

```
Define intent fwdIntent:
  do action('route')
  for target('traffic')
  from endpoint('h1')
  to endpoint('h7')
  following path ('s1',
      's8', 's12')
```

(b) Intent expression using NILE.

```
{api_key: "test_key",
 routes:[{
    key: "intent1",
    route: [
      "00:00:00:00:00:01/None",
      "of:0000000000000001",
      "of:000000000000000c",
      "00:00:00:00:00:07/None"]
    }]}
```

(c) Intent deployment using the ONOS RestAPI.

Fig. 3: An example of a network intent transformation from human language to a NILE language statement and an ONOS API call.

since they increase the cognitive load. In parallel, domain-specific language (DSL) elements have a risk of users ultimately causing a misconfiguration. Programming requires high levels of precision and programming languages exhibit more structure in comparison to natural language. Inexperienced users find it difficult to transform the mental plan of an intent into the required level of precision of the intent DSL [13] and can introduce inaccurate expressions.

In order to motivate the discussion on the need for richer intent primitives, for the rest of this section we discuss the requirements of two business intents for ISPs and describe a series of user challenges.

a) Traffic rerouting: Traffic rerouting is a common process within ISPs. Such requests can be triggered by planned maintenance tasks, *e.g.* route upgrade, or as a result of exceptional circumstances, *e.g.* a link failure. Although recent technologies, like MPLS-TE or PCE-based TE, provide automatic rerouting for some exceptional circumstances, several maintenance tasks require human intervention to drain traffic from a link, or even manually configuring an alternative path. In parallel, ISPs must specify future maintenance windows and plan in 'pen and paper' the impact of the re-routing process. During planned maintenance, re-routing will normally be applied for a longer period than the actual duration of the upgrade, in order to factor in any possible delays. Furthermore, because rerouting is a manual process, maintenance windows must be planned ahead of time and may be affected by external factors, such as hardware delivery times.

An intent system can significantly reduce the impact of an upgrade to service delivery. Infrastructure engineers can signal the start and the end of the reroute intent, without the need for a network engineer to plan and apply the operation. Nonetheless, the user must specify several details for the reroute intent, including the details of the affected devices and the severity of the re-route intent, *e.g.* should the system aim to fulfill QoS guarantees.

b) Service protection: A major event, such as the Olympic Games, can have a potentially dramatic impact on the traffic observed across network infrastructures. During such events, network operators will typically avoid making any network changes, to reduce the likelihood of an impact on their service whilst a high-profile event is underway. These are known as "service protection periods". Currently, ISPs rely on quite simple mechanisms, such as e-mail and mailing lists, to announce such protection periods within their organization. It is the responsibility of network managers, during maintenance planning, to check the relevant e-mail mail lists and ensure that no conflicts occur.

Intents have the potential to be of significant benefit, by enabling the automatic enforcement of such operations. Local managers can input service protection intents, that postpone any network changes by other intents. These intents are extremely useful for non-technical local managers who lack the necessary technical expertise to enforce such policies. Such intents require the specification of several parameters, including the scope of their policy, and require mechanisms that can describe the impact of the policy change.

IV. Architecture

Our architecture aims to fulfill two primary goals. We want to develop support for new management intents, beyond connectivity, and to explore novel mechanisms to capture intents from non-technical users. To meet these goals we have adopted a multistage translation process, supported by an intermediate intent model. The intermediate intent model remains human-friendly and ensures sufficient structure that allows easy translation into a set of low-level policies [14].The system consists of three service, depicted in Figure 2: (i) the *Interface*, which performs the initial analysis of the user input and communicates feedback on the intent state, (ii) the *Translation*, which is responsible to validate and translate user input into the intermediate representation and to generate user feedback, and (iii) the *Build*, which managed the network infrastructure and translates NILE intents into the network configuration.

The process is initialized with the user describing its intent input (either by typing or over voice) to the Interface service. The user input is processed and the keywords capturing the user's intent are extracted. The extraction process identifies the required network operation (*e.g.* route), as well as, the relevant

parameter (*e.g.* network endpoints, path), and a translation process creates an intent compatible with the intermediate intent model. The resulting intent is sent back for user validation and the user can re-state its intent in order to correct any translation errors. This is the first loop in the intent life-cycle and requires human involvement. If the user accepts the mapping, the resulting intent is deployed to the network by the build process through an SDN controller, by the second control loop.

The second loop is managed by the Translation service. It validates that the processed intent represents the user requirements, via the Interface service, and ensures that the intent can be fullfilled by the Build service. Once the intent is validate, the Translation service send the intent to the Build service for deployemnt. Feedback from the build system helps the system to refine the intermediate language translation for future requests. The third loop is manage by the Build service and provides useful feedback to the user regarding the state of the intent (*e.g.* deployment errors, failures). All three loops in the life-cycle of intents are important to validate the mapping, to self-learn, and report back to the user.

Intent Processing Pipeline: Our system adopts a micro-service approach, implementing each processing stage as a standalone service, and re-uses a series of off-the-shelf software and services to realize our intent layer and the language processing functionalities. Individual services interact through a well-defined API and an example of the transformations supported by our system is depicted in Figure 3.

The Interface service is implemented using the Google Assistant platform. The service is implemented as an application in the Google Assistant ecosystem, it allows users to access intents via a wide range of devices and transmit user input to a web service for processing.

User input is processed using Dialogflow [15], an NLP service from Google. Dialogflow allows developers to seamlessly design and integrate multi-platform conversational user interfaces with web services. The platform models user interactions using flows and state machines. User interactions are modeled as flows of state machines and each state requires the extraction of specific information elements. To effectively extract the required information, the user must provide several input examples and label in each example the information that needs to be extracted. For example, the sentences "reroute traffic between h1 and h7, through s1,s2, and s12" and "redirect flows between h1 and h7, using s1, s2 and s12" should generate a JSON object with `h1` and `h7` as the path endpoints and the forwarding path should contain the devices `s1, s2,` and `s12`. Using this dataset, Dialogflow trains an extraction model to automate input parsing, specific to the application. Finally, we configure the Dialogflow service to forward output data to the Translation service.

The Translation service of our system transforms the JSON data from Dialogflow into an intent representation. Furthermore, the Translation service validates the intent parameters and notifies the user via Google Assistant if the requested intent cannot be fulfilled (*e.g.* nodes do not exist, path is not possible). We employ an intent definition language that extends the syntax of the NILE language [11] to represent intents. NILE is an intent language designed to capture connectivity and middlebox intents and offers a human-friendly syntax. Our system extends the syntax of the language and adds support for new actions (*i.e.*, service protection, reroute) and intent management statements. Specifically, we extended the compiler to parse intent statements that contain `action` and `following` statements. Actions allow users to manage installed intents (*e.g.* re-route, protect), while following allows users to specify path preferences when defining connectivity intents. Because the NILE language remains human-friendly and rather high-level, extending the parser was easy and involved minor modifications in the language specifications. The Translation service runs as a Heroku instance [16]. An example output of the translation process is depicted in Listing 3b, when processing the output from Figure 3a.

Finally, the Build service of our pipeline translates NILE intents into SDN NBI (ONOS) RestAPI calls. Our implementation uses a Heroku instance running a Python service and uses PLY(Python Lex-Yacc) [17], a Python Lex and Yacc parser, to automate the translation. For the sake of this paper, the deployment is mainly done through the ONOS controller [18]. Listing 3c depictes an example API call to ONOS, that fullfils a re-route intent.

Due to our architecture focus, our prototype implementation offers some intent validation and optimization functionalities. The Build service performs basic connectivity and endpoint name validations, while the intent deployment algorithm adopts a first-fit approach. If an intent cannot be fulfilled, then appropriate feedback will be sent to the user via the Interface service. The modular design of our system allows easy integration of advanced optimization algorithms and we aim to explore as future work how to connect the user with planning and validation decisions.

V. Evaluation

In this section, we demonstrate the capabilities of our intent system on network management. Initially, we demonstrate the ability of our system to interact with users and manage a large emulated network. Furthermore, we evaluation the impact of our implementation on management latency, using two representative intent use-cases. As future work, we plan to design user studies to further understand the user-friendliness of the system interface and understand its impact on the user cognitive load.

s

A. Experimental setup

Due to the micro-service architecture of our system, our testbed is distributed across three location. The Interface service runs in a Google cloud instance. The Translation and the Build services run on remote Heroku cloud instances. Finally, we run a Mininet instance in a server in our local datacenter, to emulate the topology of a large ISP infrastructure. The

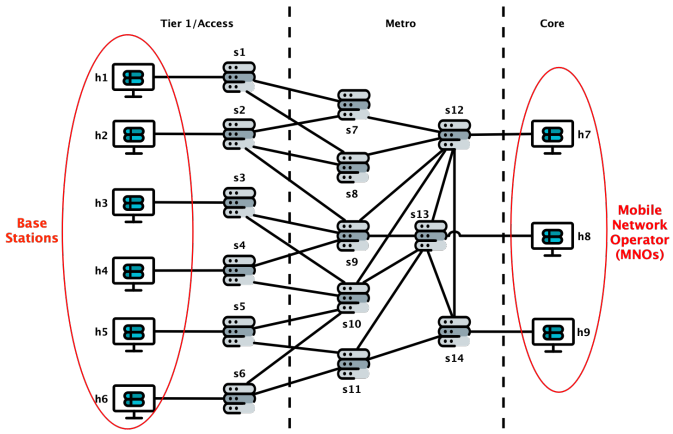


Fig. 4: Simplified topology of the BT 21CN network

TABLE I: Experimental Environment.

Setup	Details/Version
Server DELL	Intel(R) Xeon(R) Silver 4110 CPU 2.10GHz, 32GB DDR4 RAM
Mininet	Version: 2.3.0d6
ONOS	Version: v1.15
Open vSwitch	Version: 2.9.8

distributed design of our testbed emulates realistically propagation latencies in a real-world scenario. Table I summarizes different details and versions of our setup.

Our topology follows the architecture of the 21CN BT network, depicted in Figure 4. The 21CN network is organized in five network layers: the Premise, the Access, the Metro, the Core, and the iNode layers. The Core layer uses a full mesh topology, to allow inter-connectivity between all areas of BT's network. The Metro layer is responsible for inter-connectivity between nodes in a smaller geographical region, such as a city, and is responsible for connectivity between the Core section and Access layers. There are often multiple paths between access nodes and the core for redundancy. Finally, the access layer connects customers to BT's network. In our evaluation, we use Mininet and OpenVSwitch, while all switches in the access, metro, and core network (from s1 to s12) are controlled by a single ONOS instance.

To demonstrate the capabilities of our system we emulate two intent scenarios: (i) a Service Protection Period (SPP) intent, which ensures that no upgrade or maintenance is going to be applied on a certain connectivity intent for a specific period, and (ii) a traffic engineering intent, where the user can ask for traffic rerouting between two nodes.

B. Intent Use-cases

As shown in Section IV, the intent goes through different stages during its life-cycle. In this section, we measure the processing latency of each stage in the context of two different use cases (discussed in Section III). For both use cases, the intent goes through the same stage sequence (Figure 5):

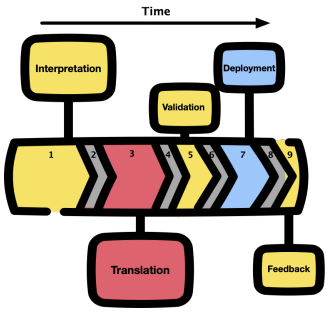


Fig. 5: Intent Execution Timeline.

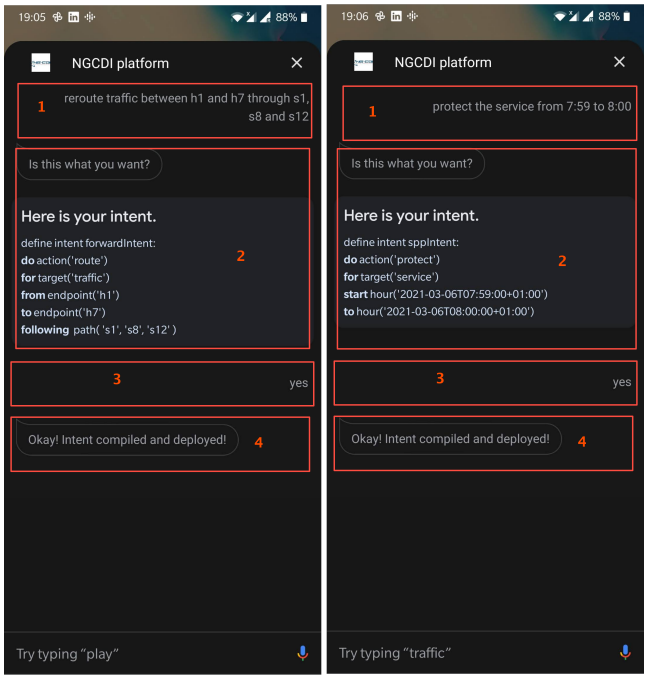
TABLE II: Intent completion time in the traffic engineering use case.

Stages	Delay	Module
Stage 1	384.4ms	Interface
Stage 2	239.4ms	Communication
Stage 3	1.05ms	Translate
Stage 4	98.35ms	Communication
Stage 5	397.6ms	Interface
Stage 6	313.43ms	Communication
Stage 7	3157.55ms	Build
Stage 8	109.22ms	Communication
Total	4701ms	N/A

1) The input intent gets dissected to keywords by the Interface service. Figure 3a shows an example of this stage in the case of traffic engineering.
2) The keywords are sent to the Translation service.
3) The Translation service transforms the user input into a NILE intent. The result will look like the example in Listing 3b.
4) A detailed description of the translated intent is sent back to the Interface service for user validation.
5) The user can validate if the generated intent matches its input.
6) If the user validates the generated intent, then a NILE intent is sent to the Build service.
7) The Build service parses the intent and translates it into a low-level configuration for deployment (ONOS NBI API).
8) Once the configuration is successfully installed, The Build service generates a notification that is printed on the user's screen by the Interface service.

Please note that all gray areas in Figure 5 represent the necessary delay for communication between each module. We refer to it as communication delay.

Figure 6 shows the Google assistant interface where the user can interact with the system in both use cases. We can see the different exchanges between the user and the system. The interaction is initiated by the user expressing its intent using either the keyboard or the microphone (Step 1). The user input is processed by the Interface and the Translation services and the resulting interpreted intent expressed in the NILE language is printing on the screen (Step 2). It is worth noting that this process exhibits a significant level of ambiguity and the system relies on the user to detect and correct it. The



Traffic Engineering **SPP**

Fig. 6: User interface showing interactions with the system

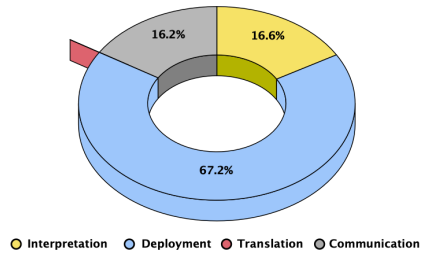


Fig. 7: Breakdown of the percentage of the total time spent in each service stage for a typical traffic engineering intent use-case.

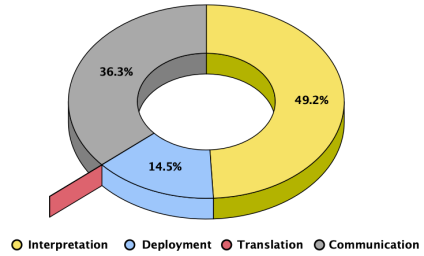


Fig. 8: Breakdown of the percentage of the total time spent in each service stage for a typical service protection period intent use-case.

user can confirm or discard the resulting intent and generate a new one (Step 3). The final intent is sent to the Build service for deployment and relevant feedback is printing on the screen (Step 4).

C. Management Overhead Evaluation

The adoption of a multi-stage translation pipeline can increase the latency of several management tasks. In this section, we provide a detailed analysis of the latencies incurred on each stage and evaluate the responsiveness of the system. Table II shows the average delays in the traffic engineering use case. From the results, we identify that stage 3 (translation) and stage 7 (build) are respectively the less and most time-consuming stages. Typically, the translation is quite a forward process, where the extracted keywords get mapped to an intermediate language. Deployment, however, takes more time as it needs to communicate with the SDN controller. Stage 2 and Stage 6 represent the average necessary delay (240ms $\approx$ 300ms) between the Interface service , hosted by Google cloud, and the Translation and the Build service, both hosted by Heroku cloud. Stage 4 and Stage 8 represent the reverse path delay (100ms $\approx$ 110ms). The difference between these latencies can be attributed to the fact that the Google infrastructure provides better connectivity, in comparison to Heroku.

Figures 7 and 8 represent the percentage of total time spent on each service stage for the Traffic engineering and the SPP intent use-cases, respectively. From the results, we highlight that for both intents the delay spent in the Translation stage is negligible in comparison to the other stages. We also highlight the different latency profiles between the two intents. In the traffic engineering intent, the Build stage takes 67.2% of the total time, while in the service protection period, the respective stage represents only 14.5% of the overall intent completion time.

To better understand the latency differences between the two intents, we compare in Figure 9 the overall intent completion time for both use cases. The SPP use case took an average of 2.48s compared with the traffic engineering use case (4.70s) to fulfill the intent. The major difference between the results is the time spent deploying the new network configuration. In our implementation, the deployment consists of parsing and transforming the intermediate language to the ONOS API. In the SPP use case, the ONOS API is a simple rule that blocks any new intent during the service protection period. However, in the traffic engineering use case, the deployment module has to wait for the SDN controller to apply the new forwarding rule in every switch of the path.

VI. Conclusion

IBN systems can drastically scale the management complexity of modern network infrastructures. To increase the adoption of IBN in production, an intent framework must consider the expression of network intents beyond matters of network connectivity alone. To cope with the semantics introduced by new intent primitives and to make platforms accessible to non-technical users, new interfaces are required that allow users to interact with the network configuration and express management requirements in a user-friendly manner. In this paper, we introduced an architecture for an IBN

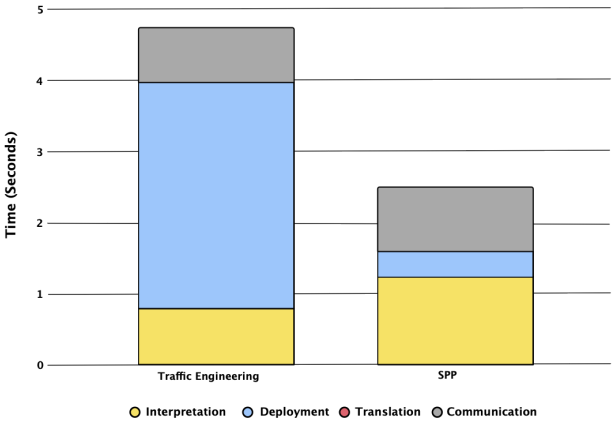


Fig. 9: Comparison of intent completion time: Traffic engineering vs. Service protection period.

framework capable of capturing new intent primitives (*i.e.*, path rerouting, service protection), and applying them across an SDN infrastructure.

For our future work, we aim to add support for additional business intents in our architecture. In parallel, we plan to explore how our system can accommodate automation in the validation and optimization of the deployment of business intents. Additional services in our pipeline, offering integration with the telemetry system, can provide run-time intent invariant validation and dynamic intent adaptation. In parallel, we aim to explore how users can be included in the process of intent validation and re-planning, either by describing invariant relaxation preferences, or break ties when multiple solutions are selected from the optimization algorithm. We believe that new user interfaces for intent systems will provide new opportunities to put the user-in-the-loop and explore intent policies from a novel perspective. Finally, designing user-friendly and intuitive interfaces is crucial for the wider adoption of intent frameworks and ambiguity is a major challenge, both in terms of analyzing the user input, as well as communicating feedback from our system. Using our intent system, we plan to conduct a series of user studies targeting a wide range of users (*e.g.* network managers, end-users, non-technical users), to understand how users process network information and define a set of design best practices.

VII. Acknowledgment

The authors gratefully acknowledge the support of the Next Generation Converged Digital Infrastructure (NG-CDI) Prosperity Partnership project funded by UK's EPSRC and British Telecom plc.

References

[1] M. Bezahaf, D. Hutchison, D. King, and N. Race, "Internet evolution: Critical issues," *IEEE Internet Computing*, vol. 24, no. 4, pp. 5–14, 2020.

[2] L. Pang, C. Yang, D. Chen, Y. Song, and M. Guizani, "A survey on intent-driven networks," *IEEE Access*, vol. 8, pp. 22 862–22 873, 2020.

[3] 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," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 29–42. [Online]. Available: https://doi.org/10.1145/2785956.2787506

[4] R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, and N. Foster, "Merlin: A language for provisioning network resources," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 213–226. [Online]. Available: https://doi.org/10.1145/2674005.2674989

[5] V. Heorhiadi, M. K. Reiter, and V. Sekar, "Simplifying software-defined network optimization using SOL," in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, ser. NSDI'16. USA: USENIX Association, 2016, p. 223–237.

[6] A. Mercian, F. Yrineu, J.-M. Kang, R. Amorim, S. M. Mahajani, M. Sanchez, and S. Banerjee, "Network Intent Composition (NIC) Feature Update and Demo: Intent Compilation, Lifecycle Management and Automated Mapping," http://sched.co/7RBY, 2016, presented in OpenDaylight Summit 2016.

[7] A. Clemm, L. Ciavaglia, L. Z. Granville, and J. Tantsura, "Intent-Based Networking - Concepts and Definitions," Internet Engineering Task Force, Internet-Draft draft-irtf-nmrg-ibn-concepts-definitions-03, Feb. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-nmrg-ibn-concepts-definitions-03

[8] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 328–341. [Online]. Available: https://doi.org/10.1145/2934872.2934909

[9] Y.-W. E. Sung, X. Tie, S. H. Wong, and H. Zeng, "Robotron: Top-down network management at facebook scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 426–439. [Online]. Available: https://doi.org/10.1145/2934872.2934874

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

[11] 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*, ser. SelfDN 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 15–21. [Online]. Available: https://doi.org/10.1145/3229584.3229590

[12] M. Bezahaf, M. P. Hernandez, L. Bardwell, E. Davies, M. Broadbent, D. King, and D. Hutchison, "Self-generated intent-based system," in *2019 10th International Conference on Networks of the Future (NoF)*, 2019, pp. 138–140.

[13] J. F. Pane, C. Ratanamahatana, and B. A. Myers, "Studying the language and structure in non-programmers' solutions to programming problems," *International Journal of Human-Computer Studies*, vol. 54, no. 2, pp. 237–264, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1071581900904105

[14] S. Alalmaei, Y. Elkhatib, M. Bezahaf, M. Broadbent, and N. Race, "Sdn heading north: Towards a declarative intent-based northbound interface," in *2020 16th International Conference on Network and Service Management (CNSM)*, 2020, pp. 1–5.

[15] Google, "Dialogflow - Lifelike conversational AI with state-of-the-art virtual agents," https://dialogflow.cloud.google.com.

[16] Salesforce, "Heroku: Cloud Application Platform," https://www.heroku.com.

[17] D. Beazley, "PLY (Python Lex-Yacc)," https://www.dabeaz.com/ply.

[18] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1–6. [Online]. Available: https://doi.org/10.1145/2620728.2620744