MD-IDN: Multi-Domain Intent-Driven Networking in Software-Defined Infrastructures

Saeed Arezoumand, Kristina Dzeparoska, Hadi Bannazadeh, and Alberto Leon-Garcia
Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON, M5S 3G4, Canada
Email: {s.arezoumand, kristina.dzeparoska}@mail.utoronto.ca, {hadi.bannazadeh, alberto.leongarcia}@utoronto.ca

***Abstract*—Intent-Driven Networking is recently gaining interest, with all major SDN control platforms now providing an intent Northbound Interface (NBI) as a high-level abstraction for network management. With these frameworks network operators can conveniently define "what needs to be done", rather than "how it should be done". Current IDN frameworks pose two main limitations that affect deployment in production grade and multi-domain networks. They are mainly concerned with a single network domain, and thus enabling end-to-end network intents over a multi-domain and large-scale setup is still a challenge. Furthermore, these frameworks do not consider any differentiation between user intents and provider intents, and a limited set of intent classes are available for both. In this paper we present MD-IDN, which provides an intent framework for the users of multi-domain cloud infrastructures. We first propose a graph-based abstraction model for user-defined intents and a generic intent compilation process. Then, we propose compilation algorithms to achieve scalability in multi-domain networks: First, user-defined intents get processed over an abstracted multi-graph of network domains and their interconnections, and a set of local intents will be generated for each of the involved domains. Afterwards, the local intents will be compiled and installed in local regions in parallel. MD-IDN is deployed as a public service in the SAVI Testbed over more than ten data centers spanning across Canada. In multi-domain environments, our experiments show that MD-IDN outperforms current practices that compile intents over a flat network topology.**

I. Introduction

With the introduction of Software-Defined Infrastructures (SDI) [1], many projects have been trying to realize SDI by combining cloud controllers (e.g. OpenStack [2]) with SDN controllers to enable network programmability for cloud users [3], [4]. During five years of continuous development and operation of the SAVI Testbed [5], a nation-wide deployment of our proposed SDI architecture, we determined that it is practically inconvenient and error-prone for most users to program their networks using low-level interfaces such as the OpenFlow protocol. Our experience confirms that realizing the capabilities of programmable networks [6] is not achievable, unless higher-level abstractions are provided for end-users. Intent-Driven Networking (IDN) promises to fill this gap by providing a simple, yet expressive high-level abstraction over the network controller [7]. This abstraction hides the unnecessary details of the underlying infrastructure from users and allows them to customize network configuration using human readable intents.

Current intent NBIs [8]–[10] compile intents over a flat non-abstracted topology, which is not scalable and feasible in multi-domain scenarios. However, a proper IDN framework for multi-domain SDIs must address certain requirements that pertain in particular to multi-tenant geo-distributed cloud environments:

Multi-domain Scale: The existing intent frameworks are not designed for multi-domain geographically-distributed SDN deployments (e.g. SAVI Testbed or Google B4 [11]). In these environments each domain has an autonomous local controller to meet the control plane response time requirements in the local network. An intent framework for these environments must install and maintain end-to-end network intents over multiple domains and hence over multiple control platforms.

Data-path Performance: Due to data-path performance requirements, these configurations cannot be applied using encapsulated overlay tunnels over IP. For example, the SAVI Testbed is comprised of data-path elements with up to 10 or even 100 Gbps of bandwidth. Data-path performance of encapsulated overlay tunnels falls far below this requirement.

Tenant Isolation: Isolation across tenants is a crucial requirement in multi-tenant environments. Therefore, the intent framework must avoid cross-contamination of intents requested by different tenants.

In this paper, we introduce MD-IDN, a framework for end-to-end Multi-Domain Intent-Driven Networking in SDI deployments. MD-IDN introduces the following particular contributions:

- A generic and extensible graph representation for user-defined network policies and intents. This intent graph abstracts away details of the network topology from users' perspective.
- We introduce and evaluate a set of algorithms to automatically distribute and scale the compilation and installation of intents in the form of an intent graph over heterogeneous and multi-domain networks.
- Our proposal for MD-IDN goes beyond a paper design. It is deployed and available as a public service for SAVI Testbed users, and has been under continuous improvement and development over the past year.

The rest of this paper is organized as follows: Section II provides an overview of the intra-domain network intent model and its characteristics. Next, section III presents the end-to-end network intent concept and its realization with the proposed

978-3-901882-98-2

algorithm. The implementation is covered within section IV. Section V proceeds with the Testbed deployment and the currently available intent classes. Section VI presents the performance evaluation and results obtained from the conducted experiments in the Testbed. Related work is summarized in section VII. Finally, conclusions and future work are presented in section VIII.

II. Intra-domain Intents

In this section we define a uniform graph-based abstraction model as a precise description of intents. As a result of this uniform foundation, a generic compilation process for user intents is provided. This uniformity allows us to decouple the intent compilation and installation process apart from the intent type and network controller platform.

A. Uniform Intent Abstraction

The technical architecture of the internal network control is usually diversified among different network domains. To build upon a uniform foundation, a simplistic graph-based representation for network intents is proposed. This abstraction model is user-centric, meaning that it is based on user requirements over the endpoints.

1) Generic Network Policies: We have two types of generic network policies regarding the endpoints in the network: Forwarding and Blocking. Forwarding can be expressed in the form of $n_1 \overset{f}{\rightarrow} n_2$, where the edge label f defines the traffic type. Additionally, a blocking intent can be defined as $n_1 \overset{f}{\nrightarrow} n_2$. As an example, to specify a forwarding intent for layer 2 traffic, the intent can be defined as $n_1 \overset{eth}{\rightarrow} n_2$. Also, to block layer 3 communication from one node to the other, the intent can be specified in the form of $n_1 \overset{ip}{\nrightarrow} n_2$.

2) Intent Graph: An intent graph is a collection of generic network policies in the form of a directed graph representing the desired policies between endpoints, without any involvement of the underlying topology. A sample intent graph is shown in Figure 1, where the policies include a bi-directional layer 2 communication among e1 and e2, a one way layer 3 connection from e1 to the gateway, and the returning path from the gateway to e1 redirected through a middlebox.

3) Mapping Functions: In order to define intent classes, a mapping function is needed to map the inputs of that specific intent class to an adequate intent graph. Once the mapping is done, the compilation, installation and the life cycle follow a uniform process. This design enables adding new intent classes as simple as writing the mapping function only. In contrast to this uniform design, within current IDN frameworks each new intent class should have a specific compiler.

B. Intent Graph Compilation

Each request to the IDN framework will be submitted to the compilation process in the form of an intent graph. This process involves three main steps:

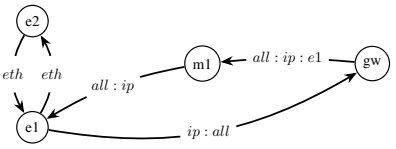


Fig. 1: Sample intent graph. e1 and e2 are endpoints (e.g. VM), m1 is a middlebox and gw is the Internet gateway.

Authorization: Upon receiving a new PG input, an examination based on the endpoint ownership information supplied by a reference monitor is completed. Since the intent graph only includes the involved endpoints and abstracts away the topology, the authorization process becomes quite straightforward. For each intent edge, the source and destination endpoints must be checked against the user's ownership and privileges. In current practices [12] where tenants have to directly define low-level flow entries to control their networks, every new flow-entry in every switch along the path has to be authorized. This enforces a considerable amount of unnecessary workload and complexity to the authorization process. In contrast, using an intent graph abstraction, the authorization process is decoupled from the network topology and flow-entries and instead it is precisely focused on the endpoints.

Feasibility Check: In the second step of the compilation process, the input intent graph gets checked by the existing intent graph for the user to ensure that the new setup does not have any conflict with the previously installed intents. In case of conflict, the user has the ability to force the intent installation.

Flow Entry Generation: After the authorization and feasibility check, the main compilation process starts. During this process, for each intent edge in the intent graph, the compiler inquires for a path between the source and destination nodes from the SDI manager and generates flow-entries based on the traffic type specified on the intent edge. This part could be used by providers to enforce their intents, as different providers may use different path selection algorithms or topology views based on their intents.

III. End-to-End Network Intents

As mentioned before, enabling multi-domain network intents is not possible by simply adding an IDN framework for the entire global network. The challenge originates mainly from scalability issues and the fact that different domains may use various control platforms and consequently not allow direct control over their network to an external entity. To address these issues, based on the proposed intent graph model in the previous section, a hierarchical and distributed IDN design is presented in this section. In this design, a global IDN first processes the requested intents over an abstracted topology to generate a set of local intent graphs.

A. Topology Abstraction

We proceed by briefly describing the topology abstractions that are required for feasible and scalable compilation of multi-

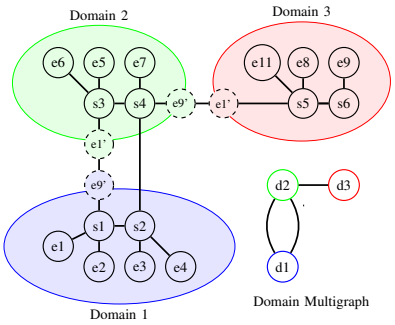


Fig. 2: A sample multi-domain topology

domain intents.

1) Local Topology: The internal network topology of each domain is referred to as local topology, shown in the collections within Figure 2. In a real multi-domain scenario where different domains could potentially have autonomous control, the domain provider would not provide the detailed local topology view to the external entities. However, the local IDN deployed over the internal control platform could have access to this topology.

2) Global Topology: The global topology is the actual detailed graph including all the nodes in each domain and the interconnections between the domains as depicted in Figure 2. It is clear that this graph grows drastically as the number or the average size of domains increases. However, this topology is not usually available in practical multi-domain environments since different domains keep their internal topologies private. In addition, no single central entity would be able to directly define control policies over such topology.

3) Inter-domain Multigraph: The Domain Multigraph is an abstracted graph where each domain is represented as a node and the domain interconnections as edges. This graph abstracts away the internal topology of the domain. This multigraph is built with minimal amount of information (only the interconnection points) provided by each domain, and thus it is much smaller than the real global topology.

B. Multi-domain Intent Compilation

The general multi-domain compilation is a two-step process. In the first step, the input intent graph gets projected over the domain multigraph resulting with the domain intent graph (Algorithm 1), which then gets processed into separate local intent graphs for each of the involved domains (Algorithm 2). Algorithm 1 receives an intent graph G_P and begins by removing all the edges whose source and destination belong to the same domain and directly places that intent edge in the local intent graph of that domain. After this process, the remaining edges in the input G_P are the ones that are between two endpoints from different domains. Next, it starts to construct the domain intent graph. To achieve that, it maps every node to its domain in the domain intent graph and for each edge in the input intent graph, an edge in the domain intent graph gets created. Figure 3 exemplifies this process, where Figure3.a shows an end-to-end Layer2 connectivity

Algorithm 1: Projection of multi-domain intent graph over domain multigraph

CompileMDIntentGraph (G_P, G_D, M)
- **inputs** : $G_P = (V_P, E_P)$ // multi-domain intent graph
  $G_D = (V_D, E_D)$ // domain multigraph
  D // mapping of nodes to domains
- **outputs** : G_dict // Dict of local intent graphs per domain
  $G_I = (V_I, E_I)$ // Projected intent graph over domain multigraph
- $initialization()$;
- **foreach** *intent edge* $e \in E_P$ **do**
  - // check if the edge is intra-domain or inter-domain
  - **if** $D[e.src] = D[e.dst]$ **then**
    - $d \leftarrow D[e.src]$
    - $G\_dict[d].addEdge(e)$
    - $G_P.removeEdge(e)$
  - **else**
    - $path_i \leftarrow getPath(G_D, D[e.src], D[e.dst])$
    - **foreach** $d_n, d_{n+1} \in path_i$ **do**
      - $f_i \leftarrow$ "$e.src : e.f : e.dst''$
      - $e_i \leftarrow newedge(d_n, f_i, d_{n+1})$
      - // create a new edge with lable of e.src:e.f:e.dst
      - $G_I.addEdge(e_i)$
- **return** G_dict, G_I;

Algorithm 2: Decomposition of domain projected intent graph into local intent graphs

CompileMDIntentGraph (G_I, G_dict, D)
- **inputs** : $G_I = (V_I, E_I)$ // Projected intent graph over domain multigraph
  G_dict // Dict of local intent graphs per domain
  D // mapping of nodes to domains
- **outputs** : G_dict // Final dict of local intent graphs per domain
- $initialization()$;
- **foreach** *inter-domain intent edge* $e_i \in E_I$ **do**
  - $(src, f, dst) \leftarrow e_i.label$
  - $dst' \leftarrow registerShadowNode(D[src], dst)$
  - $e_{scr\_domain} \leftarrow new\ edge(src, f, dst')$
  - $G\_dict[D[src]].addEdge(e_{scr\_domain})$
  - $src' \leftarrow registerShadowNode(D[dst], src)$
  - $e_{dst\_domain} \leftarrow new\ edge(src, f, dst')$
  - $G\_dict[D[dst]].addEdge(e_{dst\_domain})$
- **return** G_dict, G_I;

intent between e1 from d1 and e9 from d3. Since d1 and d3 are not directly peered, a transit domain like d2 is required. Accordingly, Algorithm 2 results with a domain intent graph as shown in Figure 3.b.

The resulting domain intent graph is delivered to Algorithm 2 which generates the adequate local intent graphs for each domain involved in the domain intent graph. Each domain can process intent graphs that only contain its internal endpoints. To identify external nodes to the internal topology of another domain, we use the notion of shadow nodes which represent mock nodes registered in the internal topology at the interconnection point. Following our example, e9 is not known for d1, thus it gets registered as e9´ in d1 and e1 gets registered as e1´ in d3. Now, both nodes are external to d2, so both e9´ and

e1´ get registered in d2. Consequently, each domain has an updated topology view and can receive a local intent graph. Figure 3.c shows the three resulting local intent graphs for each domain.

The second row of Figure 3.d shows a multi-domain service chaining intent graph in which the source is in d1, two middleboxes are in d2 and the destination is in d3. Upon running Algorithm 2, the domain intent graph shown in Figure 3.e is obtained and it gets passed to Algorithm 3 in order to generate the three local intent graphs as shown in Figure 3.f, including the required shadow nodes.

IV. Implementation

We now describe an implementation of MD-IDN as shown in Figure 4. The architecture has two main software components: the local IDN and the global IDN. Our Python-based implementation of the main platform components has over 6000 lines of code and has been under continuous refinement since last year. We proceed by briefly describing the implementation of the main components.

A. Local IDN

The local IDN provides the basic intent graph compilation for each domain. Upon receiving an input intent graph, it performs the local compilation process. To find paths between endpoints, it uses the southbound APIs provided by the local SDI manager. In addition, it may receive requests to add shadow nodes, upon which it adds the nodes to the current topology view. The local IDN also tracks events in the local network, thus if certain network events that might affect the existing intent intent graphs occur, it will trigger a recompilation based on the latest network changes and will essentially re-install the intent, if possible. Within the local IDN, each intent might have different states including pre-compilation, compilation, installation, installed and failed.

B. Global IDN

The global IDN receives multi-domain intent requests from users and maps each request to an adequate intent graph using mapping functions. It then performs a two-step compilation provided by Algorithm 1 and Algorithm 2. The resulting set of local intent graphs will be submitted to the local IDN for the involved domains. To construct and maintain the domain multi-graph, a simple domain discovery mechanism is implemented which enables the global IDN to receive domain interconnection points from each local IDN. After the compilation process, all the local intent graphs will be submitted to the local IDNs using asynchronous non-blocking RESTful calls in parallel. Once the results are received, if a failure had happened within the domains hosting an endpoint, the intent setup will fail; otherwise, if the failure happened in a transit domain, the global IDN will try to find an alternative transit domain, if one exists. Multi-domain intents can have different states similar to the local IDN states. In our primary implementation the global IDN is a single centralized platform. However, in the case of many multiple local IDNs, the global IDN could be implemented in a distributed fashion where each instance is connected to a set of local IDNs.

V. Testbed Deployment and Use cases

We have deployed MD-IDN in the SAVI Testbed, a multi-tenant Software-Defined Infrastructure (SDI) composed of heterogeneous resources including virtual machines, bare-metal, FPGA and GPU servers connected through virtual networks. Users manage their resource slices through programmatic interfaces (i.e. APIs). The Testbed is comprised of twelve regions spanning from the Postech Edge in South Korea to several other university edges in Canada, more specifically two core data-centers located in Toronto and ten other university edge regions.

To deploy MD-IDN, we have added a local IDN to each region which uses southbound APIs offered by the SDI manager to retrieve the required information and to install flow-entries to the underlying network. The required information includes the interconnection points to the other domains and the available paths between the internal endpoints based on the provided topology view by the SDI manager. The local IDN provides northbound APIs for submitting intent graphs and registering shadow nodes. Apart from local IDNs in each region, a deployed global IDN receives end-to-end intent requests from users, and it generates and submits local intent graphs for each of the involved domains.

A. Available Use cases

Over the above described deployment, we have provided five intent classes and mapping functions from intent inputs to intent graphs accordingly. Each of these intent classes enables a unique network service beyond the common networking services available in OpenStack. The intents can be easily provisioned by the Testbed users from an extended web UI portal or by using RESTful APIs. We proceed by briefly summarizing each of these classes.

Virtual L2 WAN: This intent class provisions a virtual WAN between a set of endpoints across the Testbed. Originally, there is no such service provided by an out of box installation of OpenStack.

Distributed Virtual Router: The DVR intent provisions a distributed virtual router between a selected set of endpoints from different IP subnets in different regions or projects. This intent enables a layer 3 connectivity between the existing IP addresses of the selected instances without the involvement of a central router.

Service Function Chaining: The service chaining intent enables redirection of a specific subset of the traffic through a pre-defined SFC. This intent enables multi-domain service chaining for which the current alternative approach is using overlay tunnels over the legacy IP inter-networking.

Tapping: The tapping intent class enables network traffic to be mirrored to an additionally specified instance taking a monitor role. The instance receives copies of network traffic for potential further processing, while allowing the default traffic path.

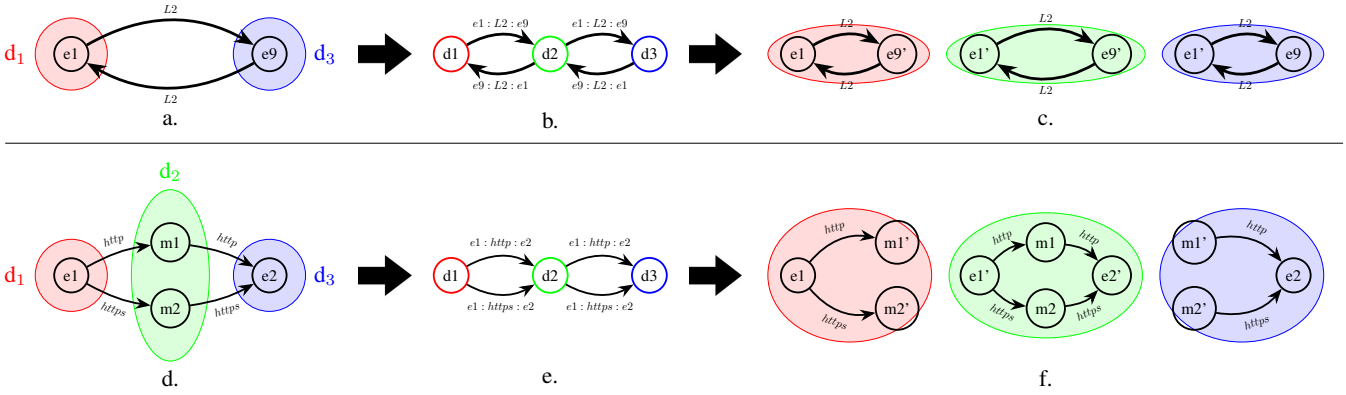


Fig. 3: Decomposition of sample multi-domain intent graphs into local intent graphs for each domain

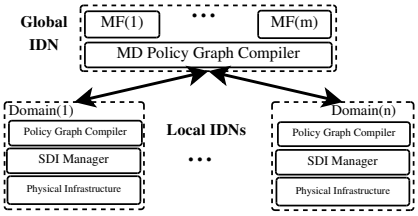


Fig. 4: MD-IDN architecture and main components

Distributed Firewall: This intent class enables multi-domain security policies in addition to the default OpenStack security groups. It operates based on a black-list approach to block malicious flows at the source domain, leaving the destination domain network uninvolved.

VI. Performance Results and Evaluation

In order to evaluate the design and implementation of the platform we have performed two sets of experiments. In the first set we targeted the performance of the multi-domain intent compilation process to evaluate the feasibility of the MD-IDN design and the effect of the proposed algorithms in a multi-domain scale. To evaluate the credibility of MD-IDN, we designed and performed a second set of experiments targeting the real use cases. In these experiments we have compared the SFC intent class as one of the new network functionalities enabled by MD-IDN with the available alternative solutions; in terms of network delay and bandwidth capacity.

A. Platform Evaluation

To evaluate our design, we measured the compilation time of an intent graph for a WAN intent over a large multi-domain topology. In these experiments two setups are compared: a flat compilation over the global topology, similar to the available alternative practices; and a hierarchical distributed compilation process of MD-IDN. To focus on the intent compilation time, we have eliminated the communication round trip times and the flow installation times.

Using the described scenario, two different experiments are performed to analyze how the MD-IDN design can improve the intent compilation performance which represents the major bottleneck as the network size grows in a multi-domain scale. In terms of path selection, CSPF [13] (Constrained Shortest Path First) algorithm is used in both cases, similar to the one used in ONOS controller [8]. In case of using optimal path selection algorithms, compilation time over the flat global topology will be considerably increased.

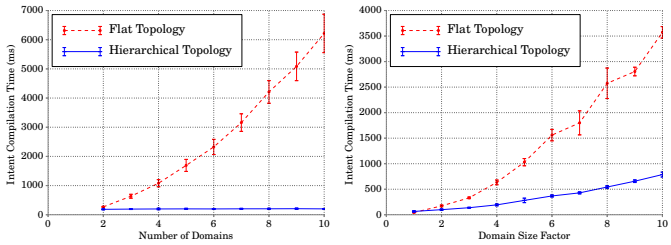
In the first experiment the compilation time is measured against an increasing number of network domains ranging from two to ten while keeping the topology size of each domain fixed with an average of 1K nodes per domain. Figure 5a shows that as the number of domains increases, the case of naive compilation over the global topology results with considerable increase, whereas the MD-IDN approach keeps the compilation time almost fixed. In case of having ten network domains, MD-IDN performs 30x faster than the naive approach. This result stems from the fact that in MD-IDN the primary compilation over the abstracted domain multi-graph happens discreetly fast, thus not contributing much to the overall compilation time. The rest of the compilation process happens in parallel in different domains. Therefore, the multi-domain intent compilation time remains almost equal to the maximum time elapsed in the local domains.

In the second experiment the number of network domains is kept fixed to four, while the size of each domain is increased from an average of 0.25K to 32K nodes per domain. As depicted in Figure 5b the MD-IDN shows a gradual increase, whereas the performance of the naive approach falls much faster.

B. Use case Evaluations

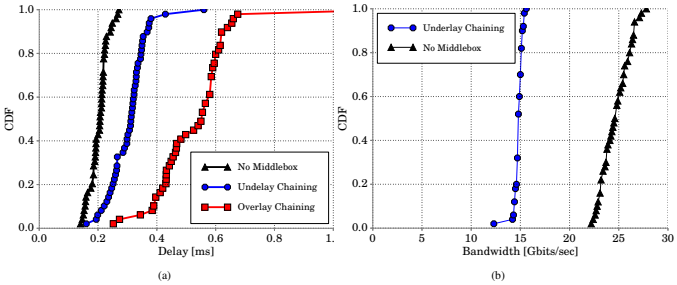
The data plane performance is evaluated through experiments based on the MD-IDN use cases that are currently available in the SAVI Testbed and we focus on the SFC intent experiment as one of the use cases.

To test the SFC intent class, we created a scenario where traffic from a source passes through two middleboxes prior to reaching the destination. As an alternative, we created the same chain using VXLAN overlays and performed delay and bandwidth tests on both. In Figure 6a, the delays incurred from



(a) (b)

Fig. 5: Intent compilation time vs global network size



(a) (b)

Fig. 6: Use case evaluation of MD-IDN in the SAVI Testbed

the chaining using VXLAN overlay, the underlay approach using MD-IDN, and the default case with no chaining involved are measured. As expected, the default one has the least delay, while with chaining included, the MD-IDN setup adds insignificant overhead, as opposed to the VXLAN chaining overhead being 3x to 4x more. In terms of the bandwidth, Figure 6b shows a comparison of the TCP bandwidth performance between the chaining intent class and the direct communication with no middlebox involved. While the default case with no middlebox saturates up to 25 Gbps, the chaining intent with two middleboxes can easily saturate up to 15 Gbps. The performance of the VXLAN case is not included in the graph, as its performance was far below the other two numbers. In fact, once the sequence of the two middleboxes is added, the TCP bandwidth drops to less than 100 Mbps.

VII. Related Work

The concept of programmable network control has recently been directed towards the definition of high level intent NBI [14], [15]. High level northbound abstractions in Software-Defined Networking is not new. Prior to the pervasiveness of intent-driven networking, many projects had tried to define intuitive and high level domain specific programming languages for SDN [16]–[18]. Even though these frameworks greatly simplify network programming, the main motivation behind IDN is to provide networking services without programming or human intervention.

In current IDN solutions [8]–[10] intents are implemented as a set of programming modules to automate the mapping between user inputs and low-level flow rules in the network. This mapping procedure (i.e. intent compilation) may vary based on the control platform, infrastructure vendor and even the intent type within the same platform. In contrast, we proposed a precise and uniform intent abstraction model based on intent graphs that can be integrated within any underlying control platform. Among the policy abstraction models that have been used in prior works [19]–[22], PGA [23] has the most similar abstraction model, where policy graphs are defined for endpoint groups in the network and are used as an intermediate abstraction for network intents. PGA targets scenarios where multiple parties should have control over one enterprise network, whereas MD-IDN mainly focuses on multi-domain and widely distributed networks.

Network topology abstraction in Software-Defined Networking has been previously proposed and used [24]–[27]. However, the unique part about our work is the use of this abstraction to simplify and scale the intent compilation process.

There are distributed or hierarchical controller designs for SDN [28]–[31], the SDI manager in the SAVI testbed [32] also follows the same approach. However, this work is not about proposing another distributed controller design. Instead, MD-IDN uses hierarchy and distribution to improve intent compilation scalability over geographically distributed network domains.

VIII. Conclusions and Future Work

In this paper we presented MD-IDN to enable end-to-end intents over multi-domain, multi-tenant networks. We started by defining a uniform intent abstraction model named intent graph which expresses extensive, new network functionalities available to users. Based on this model, a uniform compilation algorithm is introduced that could potentially consider providers' intents as well. This uniformity allowed us to extend the intent framework to cover multi-domain intents, where each domain could have autonomous network control and would only require receiving and installing local intent graphs. A set of algorithms are provided to map an intent graph spanning multiple network domains to a set of local intent graphs. The proposed architecture is implemented and deployed in the SAVI Testbed over multiple regional network domains. To demonstrate the value of MD-IDN, we developed and deployed five intent classes available to the Testbed users. Our evaluations show that MD-IDN can easily scale to numerous multi-network domains, each containing hundreds of nodes. Moreover, our use case experiments confirm that MD-IDN-enabled network functionalities provide better networking performance to Testbed users than the available alternatives.

For future work, we consider adding the role of Software-Defined Exchanges to enable multi-domain SDX intents. This work represents an initial step and provides a feasible and practical direction towards an Intent-Driven Internet, where users could request end-to-end network intents over multiple providers. We also consider extending this work to use optimal path selection heuristics in internal domain networks and design an optimally fair bandwidth allocation mechanism for the inter-domain traffic.

REFERENCES

[1] J.-M. Kang, H. Bannazadeh, H. Rahimi, T. Lin, M. Faraji, and A. Leon-Garcia, "Software-defined infrastructure and the future central office," in *Communications Workshops (ICC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 225–229.

[2] "Openstack," 2017. [Online]. Available: https://www.openstack.org/

[3] A. Al-Shabibi and L. Peterson, "Cord: Central office re-architected as a datacenter," *OpenStack Summit*, 2015.

[4] L. Peterson, S. Baker, M. De Leenheer, A. Bavier, S. Bhatia, M. Wawrzoniak, J. Nelson, and J. Hartman, "Xos: an extensible cloud operating system," in *Proceedings of the 2nd International Workshop on Software-Defined Ecosystems*. ACM, 2015, pp. 23–30.

[5] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "Savi testbed: Control and management of converged virtual ict resources," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 664–667.

[6] T. Lin, J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "Enabling sdn applications on software-defined infrastructure," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–7.

[7] C. Janz, "Intent nbi–definition and principles," *Open Networking Foundation, Version*, vol. 2, 2015.

[8] "Onos intent framework," 2017. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Intent+Framework

[9] "Opendaylight network intent composition," 2017. [Online]. Available: https://wiki.opendaylight.org/view/Project_Proposals:Network_Intent_Composition

[10] "Nemo: An applications interface to intent based networks," 2017. [Online]. Available: http://nemo-project.net/

[11] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[12] A. Gupta, N. Feamster, and L. Vanbever, "Authorizing network control at software defined internet exchange points," in *Proceedings of the Symposium on SDN Research*. ACM, 2016, p. 16.

[13] K. Manayya, "Constrained shortest path first," 2010.

[14] R. Cohen, K. Barabash, B. Rochwerger, L. Schour, D. Crisan, R. Birke, C. Minkenberg, M. Gusat, R. Recio, and V. Jain, "An intent-based approach for network virtualization," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 42–50.

[15] T. Subramanya, R. Riggio, and T. Rasheed, "Intent-based mobile backhauling for 5g networks," in *Network and Service Management (CNSM), 2016 12th International Conference on*. IEEE, 2016, pp. 348–352.

[16] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular sdn programming with pyretic," *Technical Reprot of USENIX*, 2013.

[17] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. J. Clark, "Kinetic: Verifiable dynamic network control." in *NSDI*, 2015, pp. 59–72.

[18] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying sdn programming using algorithmic policies," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 87–98, 2013.

[19] D. Ranathunga, M. Roughan, P. Kernick, and N. Falkner, "The mathematical foundations for mapping policies to network devices." in *SECRYPT*, 2016, pp. 197–206.

[20] H. X. Nguyen, M. R. Webb, and S. Naguleswaran, "Achieving policy defined networking for military operations," in *Military Communications and Information Systems Conference (MilCIS), 2016*. IEEE, 2016, pp. 1–6.

[21] L. Cui, F. P. Tso, and W. Jia, "Heterogeneous network policy enforcement in data centers," 2017.

[22] L. Cui, F. P. Tso, D. P. Pezaros, W. Jia, and W. Zhao, "Plan: Joint policy-and network-aware vm management for cloud data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1163–1175, 2017.

[23] 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.

[24] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the one big switch abstraction in software-defined networks," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 13–24.

[25] J. McCauley, Z. Liu, A. Panda, T. Koponen, B. Raghavan, J. Rexford, and S. Shenker, "Recursive sdn for carrier networks," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 4, pp. 1–7, 2016.

[26] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker, "Snap: Stateful network-wide abstractions for packet processing," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 29–43. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934892

[27] T. Soenen, S. Sahhaf, W. Tavernier, P. Sköldström, D. Colle, and M. Pickavet, "A model to select the right infrastructure abstraction for service function chaining," in *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*. IEEE, 2016, pp. 233–239.

[28] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[29] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[30] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 19–24.

[31] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–4.

[32] B. Park, T. Lin, H. Bannazadeh, and A. Leon-Garcia, "Janus: Design of a software-defined infrastructure manager and its network control architecture," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016, pp. 93–97.