Resource-Aware Intent Compilation for Virtual Private Cloud

Jia Zhang
Xi'an Jiaotong University

Wanyue Cao
Xi'an Jiaotong University

Qiang Fu
RMIT University

Hao Li
Xi'an Jiaotong University

2024 IEEE Symposium on Computers and Communications (ISCC) | 979-8-3503-5423-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISCC61673.2024.10733611

***Abstract*—Migrating enterprise IT services to the cloud is becoming a trend. However, configuring virtual networks in the cloud is a costly and error-prone task. In this paper, we model the problem of VPC intent compilation and prove that the problem is NP-hard. We design a heuristic method to automatically translate the intents into VPC configurations. To facilitate the heuristic, we propose an algorithm for finding small cuts (AFSC) based on the Louvain algorithm, which is used to separate subnets across VPCs. The generated configurations find an equilibrium between maximizing the network performance and minimizing the resource consumption. Experimental results show that our heuristic method reaches 100% correctness, compiles large intent sets from real-world networks with hundreds of subnets and thousands of intents in under a minute, and improves the effectiveness by $\sim$2.8$\times$ in terms of resource consumption relative to network performance.**

I. Introduction

Traditional enterprise IT infrastructure is bulky and difficult to expand and maintain. It has gradually become unable to meet the requirements of emerging businesses. Enterprises need a secure, flexible and lower-cost IT infrastructure. Virtual networks in the cloud can meet the needs of enterprises, providing a variety of computing and storage services. This makes the cloud a popular choice for enterprise networks.

As the most crucial component of the virtual network in the cloud, the Virtual Private Cloud (VPC) provides an isolated yet flexible network environment that can be viewed as a user's network container in the cloud. With this network container, users can deploy the cloud services they need and enjoy the rich network management functions. More importantly, this network container enables users to have complete control over their network environment and implement their physical network intents faithfully, such as subnet reachability and isolation. However, even with plenty of highly-abstracted components provided by VPC such as VPN and ACL, the manual configuration raises a series of issues.

First, constructing virtual networks in the cloud is a complex task that requires significant expertise. Since the components of VPC are just the virtual version of the physical network's low-level building blocks, their abstraction levels are not high enough to dock high-level user intents. Someone needs to complete the conversion from high-level network intents to low-level building blocks. For example, if we want to realize the isolation between two instances in the cloud, we will need to configure VPCs, subnets, and network ACLs. Second, configuring virtual networks in the cloud requires a deep learning curve. These virtual networks provide various network services, each of which requires plenty of manual configurations. For example, every rule in an ACL list requires the configuration of directions, actions, protocols, source and destination addresses, ranges of source and destination ports. This complexity may lead to errors and inefficiencies. Third, ensuring the desirable configurations of virtual networks in the cloud is a challenging task that requires the careful consideration of various factors. Since it is difficult for users to have a deep understanding of the virtual network's configuration resources, it is challenging to ensure the performance of the virtual network with the efficient use of configuration resources. For example, there are suggested constraints for some components of the VPC, to ensure the performance of the underlying physical networks. However, users may not notice or follow these constraints when they construct virtual networks in the cloud.

All these problems call for a compiling system that inputs the high-level intents of the network and outputs the low-level configurations to activate the VPC in a timely manner. The major challenge of the desired system lies in the fact that an intent can be implemented in various ways, each of which achieves different performance and consumes different resources. This raises an intuitive question: how do we configure the VPC with a low compilation latency, to ensure the correctness of the intents and the performance of the virtual network in a resource-efficient manner?

We address this challenge as follows. We first consolidate VPC intents, i.e., the reachability and the isolation of subnets. Then, we model the compilation problem as how to group subnets to minimize connection overhead between them while respecting resource quotas. As it is an NP-hard optimization problem, we propose a heuristic method to efficiently find a good solution. Our contributions are summarized below.

- We model the problem of compiling high-level intents into VPC configurations and prove it to be NP-hard.
- We propose a heuristic method to solve the problem, which translates the intents in a timely manner and ensures efficient use of resources and network performance.
- We conduct a series of experiments based on real and synthetic VPC intents. The results show that our system correctly compiles large intent sets from real-world networks with hundreds of subnets in under a minute, and improves the effectiveness by $\sim$2.8$\times$ in terms of resource consumption relative to network performance.

TABLE I
VPC BUILDING BLOCKS.

Building blocks	Functionality
Subnet	Deploy cloud resources, e.g., virtual machines
Route table	Control the traffic direction of subnets
ACL	Deny/accept traffic between subnets
Security group	Deny/accept traffic between instances
Peering connection	Connect two different VPCs
NAT gateway	Connect instance with Internet
VPN	Connect VPC with remote (physical) network

II. BACKGROUND AND PROBLEM STATEMENT

In this section, we describe the background of VPC, including its intents and building blocks, and then introduce the problem in compiling the intents into the building blocks.

A. Intents and Building Blocks in VPC

To ensure that VPC configurations accurately implement high-level intents, the connectivity of subnets needs to be determined, that is, determining whether two subnets should be isolated or reachable. While physical networks might also include waypoints (reachable through a specified router) and load balancing (reachable through multiple links), VPCs hide the implementation details of the routers and links. As a result, such intents cannot be specified in a VPC environment and are instead automatically tuned and ensured by the VPC infrastructure. In summary, current VPC intents mainly consist of the *reachability* and the *isolation* of subnets.

To compose a VPC, operators must specify three kinds of configurations: the architecture of the VPC (including subnets and the topology), the security policies of the VPC (including ACLs that secure subnets and security group policies that protect virtual machines), and connection policies that indicate how instances and multiple VPCs are connected (e.g., through a public IP, NAT gateway, or peering connection). We summarize these VPC building blocks in Table I.

B. Problem Statement in VPC Intent Compilation

Given the complexity of VPCs and to be practical, VPC intent compilation needs to be completed with a low compilation latency and ensure the correctness of the generated VPC configurations. Since the topology in a VPC is very flexible, there may be several ways to realize the intents, and each way may have different performance and consume different resources. Consider a simple reachability intent that connects two subnets. One way to implement this intent is to put these two subnets in a single VPC, which ensures reachability between them by default. Alternatively, we could put them into two individual VPCs and manually connect them by adding a peering connection between the two subnets. In this simple example, the first implementation has an obvious advantage. It does not require the additional resources for setting up the peering connection, and the connectivity inside a VPC is already there by default and yields a low latency. However, things turn around when we consider an isolation intent. While one must use ACL resources to isolate the subnets within the same VPC, the subnets in two separate VPCs are isolated by default. Note that all resources including the number of peering connections, ACLs, and security groups in a VPC have their quotas. Therefore, the goal is to find a solution that has a low compilation latency and minimizes the network latency with resource constraints.

III. VPC INTENT COMPILATION PROBLEM

In this section, we model the VPC intent compilation problem (VPC-ICP), and prove that VPC-ICP is NP-hard.

A. VPC-ICP Modelling

A network intent is reflected in the reachability and isolation relationship between subnets. In VPC, there are two options to realize such a relationship between two subnets.

- Configure the two subnets into a single VPC. This configuration ensures reachability between the two subnets by default. If the relationship between the two subnets is isolation, then configure each subnet with an ACL rule to reject the access from each other.
- Configure the two subnets into two separate VPCs. This configuration ensures isolation between the two subnets by default. If the relationship between the two subnets is reachability, then establish a peering connection between these two VPCs and add corresponding peering connection routes to each subnet.

Both options consume VPC configuration resources. In the first option (a single VPC) the cost is ACL rules, while in the second option (two separate VPCs) the cost is peering connections and routes. Recall that all these VPC configuration resources have their own quotas, and the network performance of these options may differ. Therefore, VPC-ICP can be seen as an optimization problem. Its constraints come from the requirement of *responding to intents correctly* and *meeting resource quotas*. Its objective is to *ensure network performance*.

To model VPC-ICP, we use the average ($average$) and the variance (s^2) of the latency between all pairs of reachability subnets as the indicator of network performance. We have:

$$\min average = \frac{\sum_{i=1}^{n}\sum_{k=1}^{n} c_{ik} * latency_{ik}}{\sum_{i=1}^{n}\sum_{k=1}^{n} c_{ik}}, \tag{1}$$

$$\min s^2 = \frac{\sum_{i=1}^{n}\sum_{k=1}^{n} (c_{ik} * latency_{ik} - average)^2}{\sum_{i=1}^{n}\sum_{k=1}^{n} c_{ik}}, \tag{2}$$

$$s.t. \begin{cases} \forall i \in N, \sum_{j=1}^{m} v_{ij} = 1 \\ \forall i \in N, \sum_{k=1}^{n} c_{ik}(1 - \sum_{j=1}^{m} v_{ij}v_{kj}) \leq route_{limit} \\ \sum_{j=1}^{m}\sum_{j'=1}^{m} \delta(j,j') * \frac{\sum_{i=1}^{n}\sum_{i'=1}^{n} c_{ii'} * v_{ij} * v_{i'j'}}{\sum_{i=1}^{n}\sum_{i'=1}^{n} c_{ii'} * v_{ij} * v_{i'j'}} \\ \leq 2 * pc_{limit} \\ \forall i \in N, \sum_{k=1}^{n}(1 - c_{ik})(\sum_{j=1}^{m} v_{ij}v_{kj}) \leq acl_{limit}. \end{cases} \tag{3}$$

$N = \{1, ..., n\}$ is the set of subnets. c_{ik} is a $0/1$ variable representing the relationship between subnets: if the relationship between subnets i and k is reachability then $c_{ik} = 1$, otherwise $c_{ik} = 0$. $latency_{ik}$ is a variable representing the network latency between subnet i and subnet k. The latency

is lower when the two subnets are in the same VPC than in different VPCs. $M = \{1, ..., m\}$ is the set of VPCs. v_{ij} is a $0/1$ variable representing the subnet assignment scheme: if subnet i is assigned to VPC j then $v_{ij} = 1$, otherwise $v_{ij} = 0$. $\delta(j, j')$ is a $0/1$ variable representing whether VPC j and VPC j' are the same VPC: if they are the same VPC then $\delta(j, j') = 0$, otherwise $\delta(j, j') = 1$. $route_{limit}$, pc_{limit} and acl_{limit} are VPC configuration quotas on the number of routes per route table, the number of peering connections and the number of ACL rules per ACL list, respectively.

B. NP-Hardness of VPC-ICP

The solution space of VPC-ICP is m^n, where m is the number of VPCs and n is the number of subnets in input intents. We will prove that this problem is NP-hard through reducing *multidimensional multi-choice knapsack problem* (MMKP) to this problem. First, we modify the form of the VPC-ICP model. The modified form of the model is shown as follows,

$$\min average = \sum_{i=1}^{n}\sum_{j=1}^{m}(v_{ij} * \frac{\sum_{i'=1}^{n} c_{ii'} * latency_{ii'}}{\sum_{i=1}^{n}\sum_{k=1}^{n} c_{ik}}), \quad (4)$$

$$\min s^2 = \sum_{i=1}^{n}\sum_{j=1}^{m}(v_{ij} * \frac{\sum_{i'=1}^{n}[c_{ii'} * latency_{ii'} - average]^2}{\sum_{i=1}^{n}\sum_{k=1}^{n} c_{ik}}), \quad (5)$$

$$s.t. \begin{cases} \sum_{i=1}^{n}\sum_{j=1}^{m} v_{ij} * (c_{i1}(1 - \sum_{j'=1}^{m} v_{1j'}v_{ij'})) \leq route_{limit} \\ ... \\ \sum_{i=1}^{n}\sum_{j=1}^{m} v_{ij} * (c_{in}(1 - \sum_{j'=1}^{m} v_{nj'}v_{ij'})) \leq route_{limit} \\ \sum_{i=1}^{n}\sum_{j=1}^{m} v_{ij} * ((1 - c_{i1})\sum_{j'=1}^{m} v_{1j'}v_{ij'})) \leq acl_{limit} \\ ... \\ \sum_{i=1}^{n}\sum_{j=1}^{m} v_{ij} * ((1 - c_{in})\sum_{j'=1}^{m} v_{nj'}v_{ij'})) \leq acl_{limit} \\ \sum_{j=1}^{m} v_{ij} = 1, i \in \{1, 2, ..., n\} \\ \sum_{i=1}^{n}\sum_{j=1}^{m} v_{ij} * \sum_{j'=1}^{m} \delta(j, j') * \\ \frac{\sum_{i'=1}^{n} c_{ii'} * v_{i'j'}}{\sum_{i'=1}^{n}\sum_{i''=1}^{n} c_{i'i''}v_{i'j} * v_{i''j'}} \leq 2 * pc_{limit} \\ v_{ij} \in \{0, 1\}, i \in \{1, 2, ..., n\}, j \in \{1, 2, ..., m\}. \end{cases} \quad (6)$$

We ignore the second objective in (5) for now. The latter multipliers in the first objective in (4) and all inequality constraints in (6) can be regarded as functions of $v_{i'j'}$, where $i' = 1, ..., n$; $j' = 1, ..., l_i$. Let the latter multiplier in the first objective be a constant function which is always equal to σ_{ij}. Let the latter multipliers in each inequality constraints be a constant function which is always equal to r_{ijk}. Then the model of VPC-ICP becomes the following form,

$$\max \sum_{i=1}^{n}\sum_{j=1}^{l_i} \sigma_{ij} * v_{ij}, \quad (7)$$

$$s.t. \begin{cases} \sum_{i=1}^{n}\sum_{j=1}^{l_i} r_{ijk} * v_{ij} \leq b_k, k \in \{1, 2, ..., m\} \\ \sum_{j=1}^{l_i} v_{ij} = 1, i \in \{1, 2, ..., n\} \\ v_{ij} \in \{0, 1\}, i \in \{1, 2, ..., n\}, j \in \{1, 2, ..., l_i\}. \end{cases} \quad (8)$$

This is essentially the model of MMKP. That is, VPC-ICP is at least as hard as MMKP. Since MMKP is NP-hard, VPC-ICP is also NP-hard.

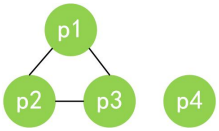

Fig. 1. Graphic representation of intents.

IV. Heuristic for VPC Intent Compilation

Given VPC-ICP being NP-hard, we now design a heuristic to solve the problem. One solution is to place all subnets included in the intents within a single VPC, and configure corresponding ACL rules for isolated subnet pairs. Since all reachable subnet pairs are in the same VPC, this scheme results in the lowest average latency and latency variance and requires no resources for peering connections or routes. However, it consumes the largest number of ACL rules to isolate subnets in the same VPC. Therefore, we start with this scheme and then adjust the distribution of subnets across VPCs to balance the consumption of ACL rules and network performance.

A. Splitting Subnets across VPCs

The adjustment is to move some of the subnets to a new VPC, and thus reduce the number of ACL rules. Ideally, we want to minimize the separation of reachable subnets and the number of ACL rules. However, this may not be achievable. Often decreasing the number of ACL rules comes at the cost of increasing the number of peering connections and routes. To this end, we stop separating reachable subnets when the quota for the number of peering connections or routes is reached.

To facilitate this process, we represent intents in a graph, where nodes are subnets and edges represent reachability relationships. Fig. 1 shows the graphic representation of the intents, where subnets $p1$, $p2$ and $p3$ can reach each other, and subnet $p4$ is isolated from $p1$, $p2$ and $p3$. The adjustment can then be seen as trading the smallest possible cut for the largest possible product of the two parts after the split into two VPCs. In other words, the adjustment will find the cut with the highest $gain_{cut}$ shown below:

$$gain_{cut} = \frac{nodenum_{part1} * nodenum_{part2} - edgenum_{cut}}{edgenum_{cut}}, \quad (9)$$

where $nodenum_{part1}$ and $nodenum_{part2}$ respectively correspond to the number of nodes in the two parts obtained by cutting the graph, and $edgenum_{cut}$ is the number of crossing edges which get cut. The nodes are then separated accordingly.

B. Algorithm for Finding Small Cuts (AFSC)

There are a tremendous number of possible cuts of a graph. It is impractical to compare all these cuts and find the exact solution. We may however only compare some relatively small cuts (in terms of the number of the crossing edges that get cut). According to (9), these cuts are likely to have a high $gain_{cut}$, among which we may select the cut with the highest $gain_{cut}$.

We propose an algorithm for finding small cuts (AFSC) based on the Louvain algorithm [1]. AFSC is divided into two

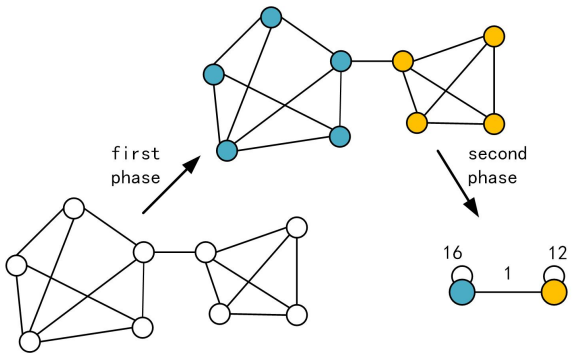

Fig. 2. Visualization of the steps of AFSC.

phases that are repeated iteratively. Assume that we start with a weighted network of N nodes. In the first phase, initially each node in the network is assigned to its own community, that is, there are as many communities as the number of the nodes. Then, for each node i we consider each of its neighbours j and evaluate the $gain_{cut}$ of the community, which is formed by removing i from its community to the community of j. Once the evaluation is completed among all its neighbours, node i is placed into the community, which generates the greatest increase of $gain_{cut}$. If there is no increase of $gain_{cut}$, i stays in its original community. This process is applied repeatedly to all nodes until no increase of $gain_{cut}$ is possible, achieving local maximum of $gain_{cut}$. The first phase is then complete.

In the second phase, the algorithm builds a new network whose nodes are now the communities from the first phase. Links between nodes of the same community lead to self-loops on this new node in the new network. The weights of the links between the new nodes are determined by the sum of the weight of the links between nodes in the corresponding two communities. Once this second phase is completed, it is then possible to reapply the first phase of the algorithm to the new network. The repetition of these two phases continues until there are no more changes in the first phase or there are only two nodes left after the second phase. Among all the communities divided at this time, the cut that separates the community with the highest $gain_{cut}$ from the rest of the communities is the cut that we are looking for. Fig. 2 visualizes the steps of AFSC, forming two communities.

C. Heuristic in Action

With AFSC, we can now identify the cuts to compare and select the one that gives the best network performance under the resource constraints. Alg. 1 shows this procedure. For each iteration of AFSC, completing the two phases and generating a new network, the best cut for the new network is identified and recorded (line 4). After AFSC completes, these best cuts are sorted according to a priority, which is defined by $gain_{bestcut} * edgenum_{bestcut}$, from the highest to the lowest (line 5). These best cuts are then evaluated to identify the one with the highest priority that meets the resource constraints (lines 6–15). This is the cut to be selected, as it gives the best performance under the resource constraints. The subnets are then split and placed into the VPCs accordingly. The VPC configurations are set up to complete the compilation.

Algorithm 1 Heuristic method for VPC-ICP

Input: Intent set
Output: VPC configurations
1: Construct the graphic representation of intents.
2: Compute connected nodes in the graph.
3: Place all the connected nodes into a group.
4: Execute AFSC. For each iteration of AFSC completing its two phases, it generates a new group. The cut with the highest $gain_{cut}$ is recorded as $bestcut$ for this group.
5: Sort all groups into a sequence by their $priority = gain_{bestcut} * edgenum_{bestcut}$.
6: **while** true **do**
7: **if** the sequence is null **or** each subnet's $num_{acl} < acl_{limit}$ **then**
8: break
9: **end if**
10: Divide the top group of the sequence into two groups according to its $bestcut$.
11: **if** after this division, $num_{pc} > pc_{limit}$ **or** $num_{route} > route_{limit}$ **then**
12: Undo this division.
13: **end if**
14: Remove the top group from the sequence.
15: **end while**
16: One group corresponds to one VPC. Configure ACL rules, peering connection and routes according to the intents.
17: Return corresponding VPC configurations of grouping.

V. Evaluation

In this section, we evaluate the proposed compiling system with real and synthetic intents and networks. We are particularly interested in answering the following questions.

- Correctness. Can the system implement the intents and generate the VPC configurations correctly, in terms of subnet reachability and isolation?
- Efficiency. Can the intent compilation complete within an acceptable time frame?
- Effectiveness. Can the system balance network performance and resource consumption?

We extract the intents from two real networks: the intents mined from Internet2 in [2] (Internet2), and the real intents from a leading VPC provider (Cloud). In addition, we randomly generate several intent sets with a various number of subnets, including an intent set for correctness checking (Correctness). Table II shows the details of these intent sets. All experiments are performed on a machine with dual 20-core Intel Xeon 2.2GHz CPUs and 192GB memory.

We compare AFSC with three representative algorithms for finding small cuts, namely, Stoer-Wagner, Karger-Stein and Louvain (§ VI). The results show that Stoer-Wagner cannot meet the resource constraints as it has the strict goal of identifying the minimum cut. In the following, we only show the comparison with Karger-Stein and Louvain.

TABLE II
DETAILS OF EXPERIMENTAL DATA.

Intent set	#Subnets	#All intents	#Reachability intents	#Isolation intents
Internet2	380	144020	7962	136058
Cloud	287	82082	59937	22145
Correctness	25	600	74	526
Random1	50	2450	286	2164
Random2	100	9900	792	9108
Random3	150	22350	1056	21294
Random4	200	39800	1784	38016
Random5	250	62250	2194	60056

A. Correctness

To verify the correctness of the intent compilation, the implementation of an intent is tested using ping. A successful ping indicates that the two subnets are reachable, representing a Reachability intent. A failed one indicates that the two subnets are isolated, representing an Isolation intent. Note that at VPC level, we are not concerned about how subnet reachability and isolation are implemented physically. We first use the Correctness intent set to verify the correctness of the intent compilation. This intent set is designed to be relatively small so that we can trace the compilation of every single intent using the method described above. The other intent sets are considerably large, making it impractical to trace all the intents. For these large sets, we perform sampling evaluation to verify the correctness of intent compilation. The results show that all the evaluated intents in each intent set get implemented (compiled), and the reachability and isolation expressions are correctly followed by the VPC configurations, giving us the confidence in the correctness of the method.

B. Efficiency

Both AFSC and Louvain have the same time complexity, as the main difference between the two is the objective. AFSC is focused on $gain_{cut}$ while Louvain is focused on community modularity. According to Alg. 1, it can be seen that AFSC is the primary source contributing to the compilation latency of the method. The time consumed by AFSC is largely determined by the community division at the bottom layer, where each original node has its own community. After rounds of community amalgamation, the number of communities or the number of nodes and edges in the network is greatly reduced, resulting in reduced computation time. The community division at the bottom layer needs to traverse all nodes in the network and the neighbouring nodes of each node. For a network with n nodes and m edges, the time complexity is $\mathbf{O}(n+m)$. In our context, the nodes are subnets and the edges are reachability intents. The time complexity of the method is therefore $\mathbf{O}(n+m)$.

Given that the time complexity is concerned with $n+m$ and indeed AFSC only deals with subnets and reachability intents, in the following we evaluate the compilation latency against $m+n$, i.e., the sum of the number of subnets n and the number of reachability intents m in an intent set. We refer to $n+m$ as the size of AFSC intent set, to distinguish from the standard intent set. Fig. 3 shows the compilation latency of the method

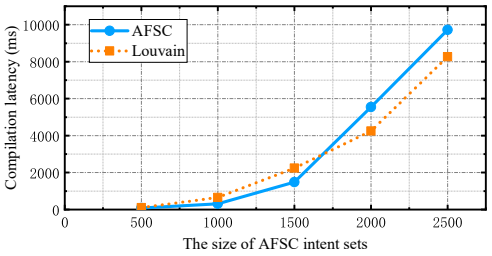

Fig. 3. Compilation latency vs. different sizes of AFSC intent set.

TABLE III
COMPILATION LATENCY FOR REAL INTENT SETS.

Algorithm	Internet2	Cloud
Karger-Stein	$>$ 24hrs	$>$ 24hrs
AFSC	5.90s	40.46s

using AFSC and Louvain, respectively, over the intent sets of Random1–5. Both algorithms have the similar performance, because they have the same time complexity. The compilation time does increase considerably, as the size of AFSC intent set grows. However, even for the largest intent set, Random5, the compilation using AFSC can be completed under 9 seconds. To have an idea of how our method performs in real-world networks, Table III shows the compilation latency over the Cloud and Internet2 intent sets between our method using AFSC and Karger-Stein [3], respectively. Karger-Stein is another popular min-cut algorithm, which can find all cuts whose weight is within a multiplicative factor of the minimum. It shows that the compilation latency of our method with AFSC is well under 1 minute, making it a practical solution. The compilation using Karger-Stein, however, cannot complete within 24 hours.

C. Effectiveness

The effectiveness is measured by the trade-off between network performance and resource consumption. Reducing the number of ACL rules is often achieved at the cost of increasing the number of peering connections or routes and network latency, and the reduction is bounded when the quota for the number of peering connections or routes is reached. Therefore, the effectiveness can be defined between the number of ACL rules and network latency as follows:

$$\frac{acl0 - acl1}{latency1 - latency0} * \frac{latency0}{acl0}, \tag{10}$$

where $acl0$ and $latency0$ are the number of ACL rules and the network latency, respectively, when all the subnets are in the same VPC, and $acl1$ and $latency1$ are the ones after the intent compilation.

Before separating subnets, the network latency of the VPC is the lowest and the number of ACL rules is the largest. The separation process is to reduce the number of ACL rules, but increase the latency, that is, the exchange of the loss in latency for the gain in ACL. The expression in (10) is essentially the gain-to-loss ratio. The larger the ratio is, the more effective the algorithm is.

Table IV presents the compilation effectiveness of our heuristic method with AFSC and Louvain, respectively. Karger-Stein is not included as the compilation using the

TABLE IV
COMPILATION EFFECTIVENESS.

Algorithm	Internet2	Cloud
Louvain	0.85	1.15
AFSC	2.18	5.04

algorithm cannot complete within 24 hours. The results show that AFSC outperforms Louvain significantly, improving effectiveness by $\sim$2.8$\times$. This is because AFSC cuts the graph with the goal of maximizing $gain_{cut}$, which finds an equilibrium between maximizing the separation of isolated subnets and minimizing the separation of reachable subnets.

VI. RELATED WORK

Network intent compilation. The existing research on network intent compilation covers both traditional networks and Software Defined Networks (SDN). In traditional networks, intents are compiled into the configurations of distributed protocols [4]–[7]. In SDN, intents are compiled into the configurations of centralized protocols or programmable devices [8]–[13]. To the best of our knowledge, there is no research on intent compilation concerning VPCs in literature. Moreover, our research has a different problem domain. All the studies above take not only intents but also network topology as input for compilation, providing their own specifications for high-level users to express their intents. Our work is, however, focused on setting up the network topology through VPC configurations based on the descriptions expressed in the intents, that is, the network topology is the output instead of the input. In addition, the compilation processes in traditional networks and SDN contain components tailored for the specific underlying implementation. Our work only concerns the characteristics of the VPC, transparent to the physical network infrastructure.

VPC-ICP modeling. Our model of VPC-ICP can be seen as a scheme choice problem. Each scheme corresponds to different costs and profits. The scheme with the highest profits is to be selected under the condition of meeting the cost constraints. This problem is similar to the knapsack problem. According to [14], our model is based on the same concept as the multidimensional multi-choice knapsack problem, which is a variant of the knapsack problem. This is also the basis to prove the complexity of VPC-ICP.

Min-cut algorithms. To find small cuts in graphs, we investigate three categories of algorithms, namely, the Stoer-Wagner algorithm [15] for finding the minimum cut, the Karger-Stein algorithm [3] for finding k-minimal cuts and the Louvain algorithm [1] for finding network communities. Experiments show that only one minimum cut for one graph cannot meet our need, as it may violate the resource constraints. For this reason, the Stoer-Wagner algorithm is eliminated. The Karger-Stein algorithm can find all cuts whose weight is within a multiplicative factor k of the minimum, but its space complexity and time complexity are both too high to be practical, as shown in the evaluation. Louvain is a community detection algorithm with the goal of maximizing community *modularity*. We borrow the process of Louvain with an optimization goal of maximizing $gain_{cut}$, needed to improve the effectiveness of our method. This modification leads to better results than Louvain from the perspective of solving VPC-ICP.

Network intent mining. The input of VPC-ICP can be either newly defined intents or intents mined from existing networks. The work in [2] shows a system called `config2spec` that automatically extracts a set of intents of a network given its configurations and a failure model. Combining our work with such network intent mining techniques can automate the process of migrating to the cloud.

VII. CONCLUSION

VPC configuration is time-consuming, tedious, and requires expertise to set up the network correctly and make it perform in a resource-efficient way. This motivates us to resort to intent compilation for VPC configuration. In this paper, we modeled the problem of VPC intent compilation and implemented a heuristic method to solve it. Our heuristic method converts in a timely manner high-level intents into VPC configurations that correctly realize the intents in virtual networks in the cloud. It ensures the efficient use of VPC configuration resources and the performance of the virtual network.

Acknowledgement. This paper is supported by the National Key Research and Development Program of China (No. 2022YFB2901403) and NSFC (No. 62172323). Hao Li is the corresponding author.

REFERENCES

[1] V. D. Blondel, J.-L. Guillaume *et al.*, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, Oct 2008.
[2] R. Birkner, D. Drachsler-Cohen *et al.*, "Config2spec: Mining network specifications from network configurations," in *Proceedings of USENIX NSDI*, 2020, pp. 969–984.
[3] D. Karger and C. Stein, "A new approach to the minimum cut problem," *Journal of the ACM*, vol. 43, no. 4, pp. 601–640, Jul 1996.
[4] R. Beckett, R. Mahajan *et al.*, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proceedings of ACM SIGCOMM*, 2016, pp. 328–341.
[5] A. El-Hassany, P. Tsankov *et al.*, "Network-wide configuration synthesis," in *Proceedings of CAV*, 2017, pp. 261–281.
[6] ——, "Netcomplete: Practical network-wide configuration synthesis with autocompletion," in *Proceedings of USENIX NSDI*, 2018, pp. 579–594.
[7] T. Schneider, R. Birkner, and L. Vanbever, "Snowcap: Synthesizing network-wide configuration updates," in *Proceedings of ACM SIGCOMM*, 2021, p. 33–49.
[8] D. Comer and A. Rastegatnia, "Osdf: An intent-based software defined network programming framework," in *Proceedings of IEEE LCN*, 2018, pp. 527–535.
[9] R. Soule, S. Basu *et al.*, "Merlin: A language for provisioning network resources," in *Proceedings of ACM CONEXT*, 2014, pp. 213–225.
[10] M. T. Arashloo, Y. Koral *et al.*, "Snap: Stateful network-wide abstractions for packet processing," in *Proceedings of ACM SIGCOMM*, 2016, pp. 29–43.
[11] M. Riftadi and F. Kuipers, "P4i/o: Intent-based networking with p4," in *Proceedings of IEEE NETSOFT*, 2019, pp. 438–443.
[12] H. Li, P. Zhang *et al.*, "A modular compiler for network programming languages," in *Proceedings of ACM CoNEXT*, 2020, p. 198–210.
[13] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "Aed: Incrementally synthesizing policy-compliant and manageable configurations," in *Proceedings of ACM CoNEXT*, 2020, p. 482–495.
[14] S. Htiouech, S. Bouamama, and R. Attia, "Using surrogate information to solve the multidimensional multi-choice knapsack problem," in *Proceedings of IEEE CEC*, 2013, pp. 2102–2107.
[15] M. Stoer and F. Wagner, "A simple min-cut algorithm," *Journal of the ACM*, vol. 44, no. 4, pp. 585–591, Jul 1997.