VITA: Virtual Network Topology-aware Southbound Message Delivery in Clouds

Luyao Luo[1,2] Gongming Zhao[1,2] Hongli Xu[1,2] Liguang Xie[3] Ying Xiong[3]
[1]School of Computer Science and Technology, University of Science and Technology of China
[2]Suzhou Institute for Advanced Research, University of Science and Technology of China
[3] Futurewei Technologies, USA

***Abstract*—Southbound message delivery from the control plane to the data plane is one of the essential issues in multi-tenant clouds. A natural method of southbound message delivery is that the control plane directly communicates with compute nodes in the data plane. However, due to the large number of compute nodes, this method may result in massive control overhead. The Message Queue (MQ) model can solve this challenge by aggregating and distributing messages to queues. Existing MQ-based solutions often perform message aggregation based on the physical network topology, which do not align with the fundamental requirements of southbound message delivery, leading to high message redundancy on compute nodes. To address this issue, we design and implement VITA, the first-of-its-kind work on virtual network topology-aware southbound message delivery. However, it is intractable to optimally deliver southbound messages according to the virtual attributes of messages. Thus, we design two algorithms, submodular-based approximation algorithm and simulated annealing-based algorithm, to solve different scenarios of the problem. Both experiment and simulation results show that VITA can reduce the total traffic amount of redundant messages by 45%-75% and reduce the control overhead by 33%-80% compared with state-of-the-art solutions.**

***Index Terms*—Southbound Message Delivery, Message Queue, Virtual Network Topology, Virtual Private Cloud.**

I. INTRODUCTION

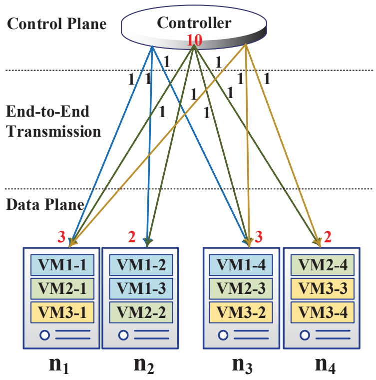
Nowadays, as more enterprise customers migrate their on-premise workloads to the cloud, the user base of a cloud provider overgrows in just a few years [1]. In current cloud deployment model, tenants deploy virtual machines (VMs) on compute nodes in the cloud data plane and manage the VMs through unified restful APIs by the cloud control plane [2]. The control plane processes tenants' requests, and sends network configuration messages, also called *southbound messages*, to computes nodes [3]. Over the past decade, we are observing rapid growth of the number of customers and the continuous expansion of individual network size. As a result, the number of southbound messages is mounting a rapid pace [4]. Thus, how to deliver the southbound messages with low provisioning latency and low control overhead has become a critical issue for hyper-scale cloud deployments [5]–[7].

A natural method to deliver southbound messages is direct end-to-end transmission via message passing interfaces (MPI) [8] or remote procedure call (RPC) [9]. For example, as one of the common protocols in distributed microservice frameworks, RPC establishes TCP links between servers and clients. In this way, each compute node directly communicates with controllers and receives all the required messages. The downside is that, as the network scale increases, the direct communication method will cause a high load on the control plane, leading to message congestion or loss, especially when encountering burst southbound traffic [10]. This insight has been discovered by the experiments [11], in which gRPC [12] and Apache Thrift [13], two widely used open-source RPC frameworks, are tested. The results show that when the payload size of each message increases from 1KB to 10KB without limitation on the sending rate, the successful queries per second drops from 10K to 4K.

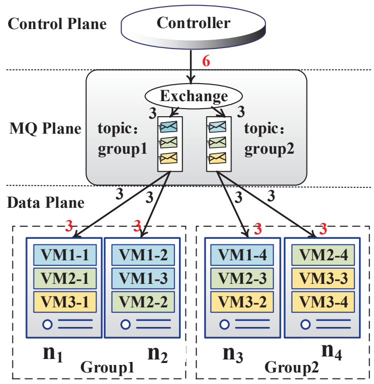
Therefore, it is necessary to reduce southbound control overhead in a large-scale cloud by decoupling the data plane from the control plane [14, 15]. As an alternative, the Message Queue (MQ) model is one of the most widely adopted messaging solutions used to build cloud infrastructure and tenant applications in the cloud [16, 17]. Specifically, a MQ server is used as a messaging middlebox between the control plane and the data plane, which implements multiple queues for storing and forwarding messages. Each queue is responsible for forwarding a set of messages with the same attributes (*e.g.*, subnet). Under this model, the controller sends messages to different queues according to message attributes, while compute nodes receive messages in one or more queues by their own needs [18, 19]. The key step in the MQ model is to *determine which queues the controller should send each message to, and which queues each compute node receives messages from.*

One of the most intuitive ideas inside the MQ model is to specify a queue for each compute node. That is, the messages are classified at the granularity of a single computing node. In this way, the control plane sends each message to an exclusive queue, and the corresponding computing node can obtain the message by subscribing to the corresponding queue. However, in reality, compared to a large number of compute nodes (such as 5,500 compute nodes in CERN [4]), a message queue server commonly supports a relatively small number of queues. For example, the experiments of Apache Kafka (a well-known open-source message queue) from [20] show that setting up a few hundred queues will lead to frequent crashes of the message queue server. Therefore, messaging at the granularity of a single compute node is not feasible in a large-scale cloud, and we must carry out message aggregation with a proper granularity.

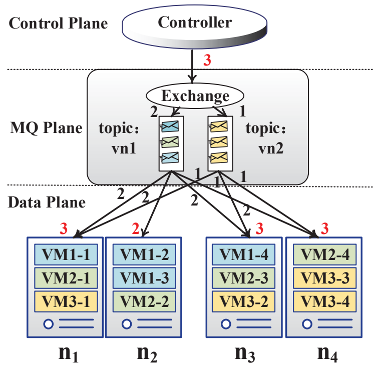
A common way for message aggregation is Node Grouping (NG) in OpenStack Nova [21]. That is, the compute nodes are



(a) Remote Procedure Calls



(b) Node Grouping



(c) VITA

Fig. 1: Illustration of interaction between controllers and compute nodes. There are one controller and four compute nodes in a cloud. VMs of three VPCs are distributed in those nodes. VM1-1, VM1-2, VM1-3 and VM1-4 belong to VPC 1. VM2-1, VM2-2, VM2-3 and VM2-4 belong to VPC 2. VM3-1, VM3-2, VM3-3 and VM3-4 belong to VPC 3. There is a message queue server containing 2 queues in the second and third subplots. The three diagrams denote three different ways of message delivery (RPC, NG and VITA).

divided into several groups, and each group of nodes shares one queue. Though this solution can reduce the number of required queues on the server, it brings a new challenge: message redundancy on each compute node. Specifically, once a compute node subscribes to one queue, it should receive all the messages from this queue to catch valid messages. Suppose that a compute node expects to receive the network configuration message m_1, and two messages m_1 and m_2 are sent to the same queue. Under this situation, the compute node will receive the redundant message m_2 because the node can only judge whether the message is valid or not after receiving it. In this way, message redundancy is inevitable. The redundant messages will occupy valuable network bandwidth and memory of compute nodes, resulting in a decrease in the overall throughput. For example, when 10,000 compute nodes are divided into 100 groups in a practical scenario, each compute node in the same group will receive the same set of messages while about 99% of messages are redundant. This will significantly reduce the resource utilization of compute nodes.

The underlying cause for high message redundancy is that NG's tight dependency on physical network topology does not align with the fundamental requirements of southbound message delivery in a multi-tenant cloud environment. That is, although VMs of a specific tenant are distributed in multiple nodes (likely across node groups), they are bounded to a logical concept called virtual networks [22]. Its implementation by cloud provider is Virtual Private Cloud (VPC) [23, 24], which is a virtual L2 overlay built on top of L3 underlay network. VPC offers isolation and privacy for tenants, and allows tenant admins to configure IP ranges, subnets, security groups, QoS policy with its boundary [23]–[25]. Therefore, it is more efficient to aggregate messages with VPC (instead of compute node) as the granularity to achieve low message redundancy.

In this paper, we design a virtual network topology-aware southbound message delivery system, called VITA. Specifically, we use VPC as the granularity to aggregate southbound messages. At the same time, considering a large number of VPCs, how to aggregate messages of these VPCs into a limited number of message queues with both low control overhead and low message redundancy is also very difficult. To solve this issue, we propose two algorithms, submodular based approximation algorithm and simulated annealing based algorithm, to solve different scenarios. Both experiment and simulation results show that VITA dramatically reduces the total traffic amount of redundant messages by 45%-75% and reduces the control overhead by 33%-80% compared with state-of-the-art solutions.

II. Motivation and VITA overview

A. A Motivation Example

This section gives an example to illustrate the pros and cons of both RPC and NG. A simple example of southbound message delivery is illustrated in Fig. 1. There are 1 controller, 4 compute nodes and 3 VPCs in the cloud. The VMs of 3 VPCs are distributed on those compute nodes. Specifically, VMs of VPC 1 are deployed on compute nodes n_1 (VM1-1), n_2 (VM1-2, VM1-3) and n_3 (VM1-4). VMs of VPC 2 are deployed on compute nodes n_1 (VM2-1), n_2 (VM2-2), n_3 (VM2-3) and $n4$ (VM2-4). VMs of VPC 3 are deployed on nodes n_1 (VM3-1), n_3 (VM3-2) and n_4 (VM3-3, VM3-4). For ease of explanation, we assume that the control plane will send a network configuration message for each VPC. The performance results are summarized in Table I.

RPC establishes connections between the controller and all compute nodes in Fig. 1(a). If a message will be sent to a VPC, the controller sends this message to the destination nodes, which contain VMs of this VPC, in turn. A mapping table is maintained in the database to record the mapping relationship between the VPCs and the compute nodes. To realize the southbound message delivery, the controller queries this table and determines compute nodes to which the messages should be sent. For example, to process the

schemes	n_1	n_2	n_3	n_4	data plane	control plane
RPC	3	2	3	2	10	10
NG	3	3	3	3	12	6
VITA	3	2	3	3	11	3

TABLE I: The number of messages received by each compute node, received by the data plane, and sent by the control plane through three delivery schemes.

configuration message of VPC 1, the controller queries the database and obtains the IP addresses of compute nodes (*i.e.*, n_1, n_2 and n_3). Then, the controller will send the configuration messages to these three nodes through RPC. As a result, the controller sends 10 messages in total and the data plane receives 10 messages accordingly.

The Node Grouping (NG) method divides the four compute nodes into two groups, as shown in Fig. 1(b), and uses a message queue server for storing and forwarding messages. All the queues are identified by topics. The controller sends message to one queue by publishing messages to a topic, and each compute node receives messages from one queue by subscribing to a topic. The MQ server in this example contains two queues, which are identified by topics $group_1$ and $group_2$, respectively. On processing the configuration message of VPC 1, the controller queries the database and obtains the nodes which require this message. The nodes n_1 and n_3 are in group 1 and group 2, separately. So, the controller should send two messages with the same content to the MQ server. One is published to topic $group_1$, and the other is to topic $group_2$. In all, the controller sends 6 messages in total. However, as the compute nodes in the same group will receive all the messages from a queue, a node will receive some invalid messages. For example, node n_2 receives 3 messages of VPCs 1, 2, and 3, but only 2 messages from VPCs 1 and 3 are necessary. Node n_4 receives 3 messages with 1 unnecessary message of VPC 1. As a result, all the compute nodes in the data plane receive 12 messages, 2 of which are unnecessary.

B. *Our Intuition*

We observe that the two solutions of southbound message delivery have advantages and disadvantages. RPC allows each compute node to receive only the required messages without any redundancy. In small-scale clouds, perhaps this is the most proper solution. However, in large-scale distributed cloud scenarios, the pressure of the control plane will be weighty, and the message delivery latency may be very high [11]. As for the node grouping solution, the pressure of the control plane can be reduced while the load on the data plane (redundant messages) significantly increases.

A question immediately following the above discussion is that *can we do better by using MQ with less redundant messages and low control overhead?* Clearly, we should use as many queues as possible for southbound message delivery. However, too many queues will lead to frequent crashes of the message queue server [20]. Therefore, how to effectively aggregate many messages into a limited number of queues is necessary. As mentioned above, southbound messages have not only physical attributes (*e.g.*, IP address of the destination node) but also virtual attributes (*e.g.*, VPC ID) under the virtual private cloud architecture. Moreover, messages from the same VPC are more likely to be sent to the same virtual address in the virtual network [26, 27]. In other words, aggregating and delivering southbound messages according to the attributes of the VPC is more intuitive and efficient than existing solutions.

As shown in Fig. 1(c), since there are 3 VPCs and 2 queues in this example, the controller aggregates the messages of VPCs 1 and 2, and sends these messages to the same queue (with topic vn_1). Meanwhile, the controller sends the messages of VPC 3 to another queue (with topic vn_2). Each compute node subscribes to different topic(s) according to the messages it needs. For example, because node n_2 only needs the messages of VPCs 1 and 2, it only subscribes to topic vn_2. Similarly, since node n_4 needs the messages of VPCs 2 and 3, it should subscribe to both topics vn_1 and vn_2. Accordingly, the controller sends 3 messages, and all the compute nodes in the data plane totally receive 11 messages, 1 out of which is unnecessary. As a result (shown in Table I), this scheme achieves lower control overhead compared with RPC, and achieves better data/control plane performance compared with NG. Motivated by this example, we design a virtual network topology-aware southbound message delivery scheme, called VITA.

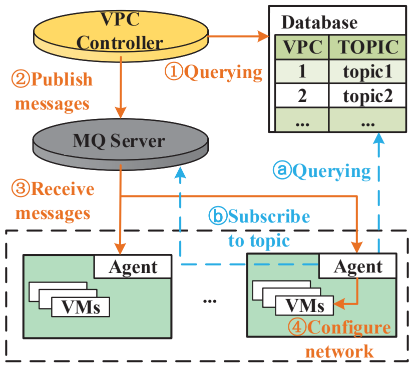


Fig. 2: Overall system overview and workflow of VITA.

C. *System Overview and Workflow of VITA*

As shown in Fig. 2, VITA mainly consists of three parts: the control plane (composed of the controllers), the data plane (composed of the compute nodes), and the message queue server. Specifically, the control plane consists of a set of distributed microservices, and one of its functions is to manage the virtual network through southbound message delivery. In order to build the correspondence between VPCs and topics, a mapping table from VPCs to topics, instead of VPCs to IPs, is maintained. We will describe in detail how to determine the correspondence in Section III. In the data plane, VMs belonging to different VPCs are distributed on different compute nodes. For more efficient implementation, a control agent is designed on each node to be responsible for subscribing to topics, distinguishing messages, and parsing requests. The agent manages all virtual machines on the node and knows to which VPCs they belong. As an important component, the MQ server is responsible for the asynchronous communication between the control and data planes.

Fig. 2 also briefly describes the system workflow. The system process is mainly triggered by two events. One is the launch of a new VM on the compute node. When a VM is added or migrated, the control agent queries the database to get the topics. Then it subscribes to those topics for receiving the required messages of different VPCs. The other one is the configurations by a tenant. When one tenant configures their VPC through provided API (*e.g.*, subnet, security group), the control plane parses the request and constructs corresponding southbound messages. Then it queries the database and determines which topic(s) the messages should be published to. Next, the controller sends the messages to specified queues and asynchronously waits for the reply of the processing result. Finally, the agent receives messages from specific queues and judges whether it is valid or not according to the VPC ID of the message. If no VM needs this message, it will be discarded. Otherwise, the control agent will perform corresponding operations (*e.g.*, setting IP, configuring routing table) on VMs according to the content of the message and return the operation result to the control plane. In this way, VITA can realize the decoupling of the control plane and the data plane.

III. Problem Formulation and Algorithm Design

A. Network Models

A typical cloud consists of the control plane and the data plane. Specifically, a cluster of controllers constitute the control plane, and are responsible for managing the network, including southbound message delivery. The data plane consists of a set of compute nodes, and is responsible for providing computing resources for tenants. We use $N = \{n_1, n_2, ..., n_{|N|}\}$ to represent the set of compute nodes. The set of VPCs in the cloud is denoted as V=$\{v_1, v_2, ..., v_{|V|}\}$. Tenants create VPCs in the cloud by deploying VMs on compute nodes.

We adopt the MQ model to implement southbound message delivery. Specifically, an MQ server containing a set of queues, serves as the messaging middlebox in a cloud and adopts the publish/subscribe model [28, 29]. The queues are responsible for storing and forwarding southbound messages from the control plane to the data plane. Each queue is identified by a topic. When the controller sends messages to one queue, we say that the controller publishes messages to the topic. The compute nodes receive messages from a queue by subscribing to the corresponding topic. The topic set is defined as $T = \{t_1, t_2, ..., t_K\}$, where $K = |T|$ is the number of queues in the MQ server.

B. Problem Formulation

The section gives the formulation of the virtual network southbound message delivery (VSMD) problem. Specifically, we use VPC as the granularity to aggregate southbound messages. Due to the prior work of traffic matrix prediction in clouds [30, 31], it is reasonable to assume that we can obtain the expected traffic intensity of southbound messages for each VPC $v \in V$, which is denoted as $f(v)$.

The key step of VSMD is to determine to which queue(s) the controllers should deliver each message, and from which queues each compute node receives messages. Thus, we use binary variable y_v^t to denote whether the controller will publish the messages of VPC v to topic t or not. Meanwhile, we use binary variable z_n^t to represent whether the compute node n will subscribe to topic t or not.

In order to deliver southbound messages successfully, we should consider the following two constraints. 1) Each compute node must obtain all the required messages. That is, each compute node should receive the messages of VPC v if a VM belonging to v is deployed on this node. The constant Γ_n^v indicates whether the compute node n contains the VMs belonging to VPC v or not. 2) The traffic amount of messages on each node should not exceed its capacity. We use $s(n)$ to denote the message processing capability of node n. Once a compute node subscribes to a topic, it will receive all the messages in this queue, which results in message redundancy. Thus, our objective is to minimize the total traffic amount on compute nodes (or in the data plane). We give the following problem definition:

$$\min \sum_{n \in N} b(n)$$

$$S.t \begin{cases} \sum_{t \in T} y_v^t \geq 1, & \forall v \in V \\ \sum_{t \in T} z_n^t y_v^t \geq \Gamma_n^v, & \forall n \in N, v \in V \\ \sum_{t \in T} \sum_{v \in V} z_n^t y_v^t f(v) = b(n), & \forall n \in N, v \in V \\ b(n) \leq s(n), & \forall n \in N \\ y_v^t, z_n^t \in \{0, 1\}, & \forall v, n, t \end{cases} \tag{1}$$

The first set of inequalities indicates that each VPC subscribes to at least one topic. The second set of inequalities represents that all the VMs on any compute node should receive all the required messages. The third set of equalities shows the message traffic amount on each compute node n, denoted as $b(n)$. The fourth set of inequalities expresses the message processing capacity constraint on each compute node n. Our objective is to minimize the total message traffic amount on compute nodes, that is, $\min \sum_{n \in N} b(n)$.

Theorem 1 *The VSMD problem is NP-hard.*

We prove the NP-hardness by showing that the Weighted Set Covering Problem (WSCP) [32] is a special case of VSMD. Due to space limit, we omit the detailed proof here.

C. Algorithm Design for VSMD

1) Algorithm Overview: If the controller sends the messages of each VPC to only one queue, the total traffic amount of messages delivered by the controller can be minimized. Considering that the controller is often the bottleneck in a cloud, it is reasonable to assume that messages of each VPC are sent to only one queue. To deal with this scenario, this section presents a submodular-based approximation algorithm to solve the VSMD problem. We will consider the scenario where the messages of each VPC can be forwarded to more than one queue in the next section.

In this section, we regard that the messages of each VPC are sent to only one queue. As a result, the VPC set can be divided into K subsets, and each VPC in the same subset is assigned with the same topic. Initially, all VPCs belong to the same set. Our algorithm consists of K iterations where K is the number of queues (*i.e.*, the number of topics) in the MQ server. In each iteration, we determine a subset of V that can reduce the total traffic amount of messages the most and assign all the VPCs in this subset with one topic.

2) Preliminaries: We first give the definition of the traffic amount of messages of VPC set $V' \subseteq V$ as follows:

Definition 1 *For any VPC set V', the total traffic amount of messages of all the VPCs in V' is*

$$R(V') = |Sub(V')| \sum_{v \in V'} f(v) \tag{2}$$

where $Sub(V')$ is the set of compute nodes which contain VMs belonging to any VPC $v \in V'$.

We need to divide the VPCs into K sets so that messages of each VPC will be published to one of K topics. Initially, when all the VPCs belong to one set, the total traffic amount of messages on all compute nodes can be expressed as $R(V) = |N| \cdot \sum_{v \in V} f(v)$, where $|N|$ is the number of compute nodes. If we divide VPCs into K sets, denoted as $\{V_1, V_2, ..., V_K\}$, the traffic amount of all southbound messages becomes $\sum_{i=1}^{K} R(V_i)$. In other words, the traffic amount of messages will be reduced as much as possible by dividing VPCs into K sets. That means the minimization problem in Eq. (1) can be converted into the following equivalent maximization problem:

$$\max R(V) - \sum_{i=1}^{K} R(V_i)$$

$$S.t \begin{cases} \sum_{t \in T} y_v^t \geq 1, & \forall v \in V \\ \sum_{t \in T} z_n^t y_v^t \geq \Gamma_n^v, & \forall n \in N, v \in V \\ \sum_{t \in T} \sum_{v \in V} z_n^t y_v^t f(v) \leq s(n), & \forall n \in N, v \in V \\ y_v^t, z_n^t \in \{0, 1\}, & \forall v, n, t \end{cases} \tag{3}$$

This problem is similar to a clustering problem, where we need to divide the VPC set V into K clusters to maximize the traffic amount reduction on compute nodes. Our algorithm is based on efficient computations of a submodular set function φ, which defines the maximum traffic amount reduction of messages by dividing the VPCs into several sets. We give the definition of the submodular set function φ as follows.

Definition 2 *Given the set Φ, which contains disjoint subsets of V, the traffic amount reduction of messages achieved by dividing the VPCs according to Φ is defined as:*

$$\varphi(\Phi) = R(V) - \sum_{S \in \Phi} R(S) - R(V - M) \tag{4}$$

where M is the set of VPCs that can be covered by all the sets in Φ. That is, $M = \bigcup_{S \in \Phi} S$.

Next, we give the definition of submodularity, and prove that the function φ is submodular.

Definition 3 *(Submodularity): Given a finite set E, a real-valued function z on the set of subsets of E is called submodular if $z(S \cup \{e\}) - z(S) \leq z(S' \cup \{e\}) - z(S')$ for all $S' \subseteq S \subseteq E$ and $e \in E - S$.*

Lemma 2 *Given the set U as the power set of V, the function φ defined in Eq.* (4) *is submodular on U.*

Proof: Without loss of generality, we consider an arbitrary set $\Phi \subseteq U$ and an arbitrary set $A \subseteq V$. Assume that A does not intersect with other sets in Φ, *i.e.*, $A \cap S = \emptyset, \forall S \in \Phi$. Then, we have

$$\varphi(\Phi \cup \{A\}) - \varphi(\Phi) = R(V - M) - R(V - M - A) - R(A) \tag{5}$$

where $M = \bigcup_{S \in \Phi} S$. Given an arbitrary subset $\Phi' \subseteq \Phi$, it also follows

$$\varphi(\Phi' \cup \{A\}) - \varphi(\Phi') = R(V - M') - R(V - M' - A) - R(A) \tag{6}$$

where $M' = \bigcup_{S \in \Phi'} S$.

Note that $R(V - M) - R(V - M - A) - R(A)$ also represents the traffic amount reduction by dividing set $V - M$ into two subsets: $V - M - A$ and A. Since Φ' is the subset of Φ, $V - M$ is the subset of $V - M'$ accordingly. Thus, we have:

$$R(V - M) - R(V - M - A) \leq R(V - M') - R(V - M' - A) \tag{7}$$

Combining Eqs. (5), (6) and (7), we know that:

$$\varphi(\Phi \cup \{A\}) - \varphi(\Phi) \leq \varphi(\Phi' \cup \{A\}) - \varphi(\Phi') \tag{8}$$

According to Definition 3, we show that the set function φ is submodular. ∎

To maintain the processing capacity constraint of a single compute node n, *i.e.*, $b(n) \leq s(n)$, we only focus on the set $A \subset V$ without breaking the constraint, that is,

$$\sum_{v \in A} f(v) \leq \min_{n \in Sub(A)} s(n) \tag{9}$$

We call the sets satisfying Eq. (9) as *feasible sets*. The feasible sets can be explored efficiently by simply performing a depth-first search [33] on the VPC set V. We omit the detailed description here due to space limit.

3) Algorithm Description: Given these insights, we propose the submodular-based southbound message delivery algorithm (SM-SMD) in detail, which is formally described in Alg. 1. SM-SMD consists of three steps. In the first step, the algorithm computes a set of feasible sets Π in advance and starts with an empty set Φ (Line 3). In the second step (Lines 5-12), it loops through the possible feasible set $S \in \Pi$ to find the maximum function value $\varphi(\Phi \cup \{S\})$. The algorithm performs $K - 1$ iterations until we obtain K sets of VPCs. In the third step (Lines 13-17), we obtain the mapping relationship between VPCs and topics (*i.e.*, y_v^t).

4) Performance Analysis: We analyze the approximation performance of our proposed algorithm based on the following lemma.

Lemma 3 *For a real-valued submodular and non-decreasing function $z(S)$ on U, the optimization problem $max_{S \subseteq U}\{z(S) : |S| \leq K, z(S)\ is\ submodular\}$ can reach a (1-1/e) approximation factor if the algorithm performs greedily [34].*

Theorem 4 *Our SM-SMD achieves a (1-1/e) approximation factor for the maximization problem in Eq.* (3).

Proof: The function φ is submodular by Lemma 2. Besides, for any set Φ of subsets of V and $A \subseteq V$ with

Algorithm 1 SM-SMD: Submodular-based Algorithm for VSMD

```
Step 1: Initialization
Compute the set of feasible sets Π
Φ ← ∅
Step 2: Greedy Selection
while |Φ| ≤ K − 1 do
    Set tmp ← 0, opt ← 0
    for S ∈ Π − Φ do
        tmp ← φ(Φ ∪ {S})
        if tmp > opt then
            opt ← tmp, S* ← S
    Φ ← Φ + {S*}
Φ ← Φ + {V − ⋃_{S∈Φ} S}
Step 3: Assignment of VPCs and Topics
i ← 1
for S ∈ Φ do
    Set y_v^{t_i} = 1 if v ∈ S
    i ← i + 1
```

$A \cap S = \emptyset, \forall S \in \Phi$, it follows $\varphi(\Phi \cup \{A\}) - \varphi(\Phi) \geq 0$, and the equal sign is held only in the case where $Sub(V - \bigcup_{S \in \Phi} S) = Sub(A)$. Thus, the function φ is non-decreasing. By Lemma 3, our proposed algorithm can reach a $(1-1/e)$ approximation factor for the VSMD problem in Eq. (3). ■

Note that the number of feasible sets may be exponential. However, the work [35] has shown that polynomial number of feasible sets are enough for performance optimization. To achieve the trade-off optimization between algorithm complexity and network performance, we only construct the polynomial number (with input the number of VPCs) of feasible sets. Under this condition, the time complexity of SM-SMD is $O(K|V|)$ since the algorithm runs in $K-1$ iterations, and the function φ is calculated $O(|V|)$ times in each iteration.

D. Simulated Annealing Algorithm for VSMD

The SM-SMD algorithm considers the scenarios where the control plane may be the bottleneck in a cloud. However, in some other scenarios, the data plane is more likely to become a bottleneck [36, 37]. Under these scenarios, we hope to reduce more traffic amount of messages in the data plane. To this end, we give a simulated annealing based southbound message delivery algorithm where southbound messages of one VPC can be sent to more than one queue (*i.e.*, $\sum_{t \in T} y_v^t \geq 1$). It should be noted that this algorithm will increase the control overhead and MQ overhead compared with SM-SMD, but reduce traffic amount on compute nodes. (*i.e.*, reduce the message redundancy).

Simulated annealing [38] is a probabilistic optimization algorithm which takes L, t_0, t_m, and α as inputs. L is the number of iterations at each temperature $\mathcal{T}$. t_0 and t_m are the initial value and the end threshold of the temperature $\mathcal{T}$, respectively. α is the decreasing rate of $\mathcal{T}$. The temperature $\mathcal{T}$ is used to determine the probability of accepting the worse state. Note that, the parameter selection of the simulated annealing algorithm has been extensively studied [38, 39]. We determine the parameters based on the work [38] to achieve a high probability for converging to the global optimal solution.

SA-SMD first initializes the parameters and the initial state. As SM-SMD can obtain a feasible assignment of VPCs and topics, SA-SMD takes the results of SM-SMD as the initial state. Then it executes a two-level iteration. In the each round of the inner iteration (Lines 4-11), the algorithm randomly selects a VPC and a topic to change their mapping relationship (*i.e.*, $y_v^t = 1 - y_v^t$) (Lines 6-7) and calculates the difference in the total traffic amount of messages on all compute nodes by re-selecting topics, denoted as Δ (Line 8). If $\Delta \leq 0$, it means that the message redundancy is reduced, and we accept the current state. Otherwise, we refuse the current state with probability $1 - e^{-\frac{\Delta}{\mathcal{T}}}$ (Lines 9-10). Each inner iteration runs in L rounds. In the outer iteration, temperature $\mathcal{T}$ is decreased by a factor α at the end of the inner iteration (Line 13). Then, if $\mathcal{T} \geq t_m$, the algorithm terminates and outputs the final result. Otherwise, it performs a new inner iteration with a decreased temperature. The SA-SMD algorithm is formally described in Alg. 2.

Algorithm 2 SA-SMD: Simulated Annealing based Algorithm for VSMD

```
1:  Input L, t_0, t_m, α
2:  Run SM-SMD to obtain the solution: y_v^t and z_n^t = Γ_n^v y_v^t
3:  Init temperature T = t_0, k = 0
4:  while T ≥ t_m do
5:      while k ≤ L do
6:          Select a random VPC v and a random topic t
7:          Set y_v^t ← 1 − y_v^t.
8:          Set Δ to be difference of total traffic amount by
            topic re-selection.
9:          if Δ > 0 then
10:             Set y_v^t ← 1 − y_v^t with probability 1 − e^{−Δ/T}
11:         k ← k + 1
12:     Set k = 0
13:     T ← αT
14: Output the results
```

In each round of the inner iteration, the algorithm calculates the difference of traffic amount received by each compute node by re-selecting topics, which costs $O(|N|)$ time. This calculation loops L times at each temperature $\mathcal{T}$, which drops from t_0 to t_m at the decreasing rate of α. Thus, we execute the calculation for $\log_\alpha(t_m/t_0)$ times and the overall time complexity of SA-SMD is $O(L \cdot \log_\alpha(t_m/t_0) \cdot |N|)$.

IV. Performance Evaluation

A. Performance Metrics and Benchmarks

This paper studies how to deliver southbound messages in clouds with low control overhead and low message redundancy. The code is open-source and available at https://github.com/futurewei-cloud/vita. We adopt five main metrics for performance evaluation. (1) *The control overhead* represents the resource consumption of the controller for southbound

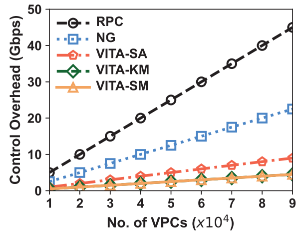


Fig. 3: Control Overhead vs. No. of VPCs

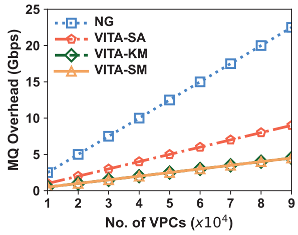


Fig. 4: MQ Overhead vs. No. of VPCs

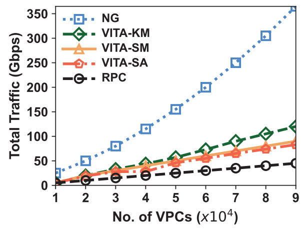


Fig. 5: Total Traffic vs. No. of VPCs

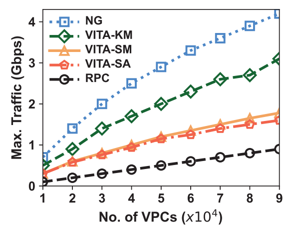


Fig. 6: Max. Traffic vs. No. of VPCs

message delivery. In the testbed experiment, we measure the controller's CPU utilization during system running as the control overhead. Meanwhile, we record the total traffic amount of messages sent by the controller as the control overhead in large-scale simulations. (2) *The MQ overhead* indicates the resource consumption of the MQ server to process southbound messages. According to [20], disk I/O utilization is the main performance bottleneck of the MQ server. Thus, we use disk I/O utilization as the MQ overhead in the testbed experiment. As for large-scale simulations, we measure the total traffic amount of the messages through the MQ server as the MQ overhead. (3) *The total traffic amount* of all compute nodes. (4) *The maximum traffic amount* of all compute nodes. We measure the total traffic amount of southbound messages received by each compute node, and calculate the total (or maximum) value of all compute nodes as the third (or the fourth) metric. (5) *The average message delivery delay*. We record the time interval from the controller sending the southbound message to the compute node receiving the message as the message delivery delay. We compute the average delivery delay of all messages during the system running as this metric.

In this paper, we propose two message aggregation and distribution algorithms, SM-SMD and SA-SMD, based on VITA. We denote the corresponding schemes as VITA-SM and VITA-SA, respectively. To evaluate the performance of our VITA-SM and VITA-SA, we choose the following three state-of-the-art solutions as benchmarks.

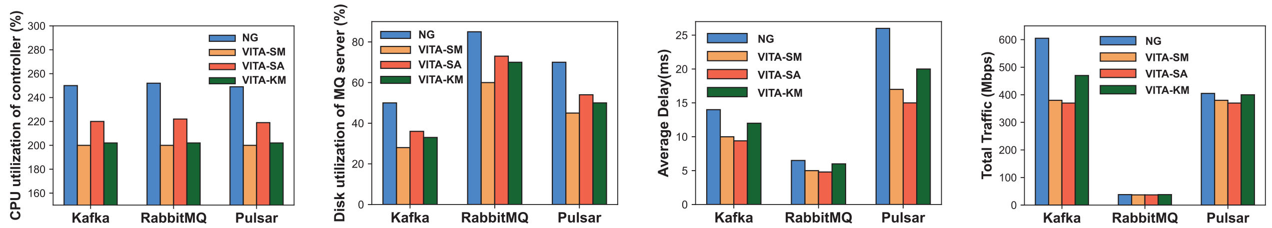
1) The first one is RPC [9], which is a widely used method in distributed microservice framework for communications between servers and clients. In clouds, RPC establishes TCP connections between the controller and all compute nodes. Messages are sent from the controller to corresponding compute nodes one by one.
2) The second one is NG [21], which performs southbound message delivery using message queues. To deal with a limited number of message queues on the server, compute nodes are divided into certain groups according to a certain attribute (such as physical location). The nodes in the same group will subscribe to a same topic (*i.e.*, queue) and receive the same messages.
3) The third one is denoted as VITA-KM. Since there is no exact work about southbound message delivery based on virtual network topology, we use the classic clustering algorithm, K-means [40], to aggregate and distribute messages with VPC as the granularity. VITA-KM takes the number of topics as the input k, and divides the set of VPCs into k clusters.

B. Simulation Evaluation

We refer to a practical private cloud deployed in CERN (European Organization for Nuclear Research) [4] to design our simulation. The CERN private cloud contains 5,500 compute nodes. We change the scale of the virtual network by varying the number of VPCs from 1×10^4 to 9×10^4. We assume that the VMs are distributed on the compute nodes randomly, and the number of topics is set to 1,100 by default. As a result, NG divides the compute nodes into 1,100 groups, and each group contains 5 compute nodes. The expected message traffic intensity for each VPC is set as 1Mbps. Moreover, we use power law for the message-size distribution, where 20% of all messages account for 80% of traffic volume as observed in [41].

We observe the control overhead, the MQ overhead, and the total/maximum traffic amount on compute nodes by changing the number of VPCs in the cloud. The results are shown in Figs. 3-6. Specifically, Fig. 3 shows that the control overhead of all solutions increases with the increasing number of VPCs, and the growth rate of VITA-based solutions is significantly slower than that of RPC and NG. For example, given 7×10^4 VPCs, the control overheads of VITA-SM, VITA-KM, and VITA-SA are 3.6Gbps, 3.6Gbps, and 6.8Gbps, respectively, while those of RPC and NG are 35.1Gbps and 17.5Gbps, respectively. It means that VITA-based solutions can reduce the control overhead by over 80% and 60% compared with RPC and NG, respectively. That is because the more messages delivered by the controller, the higher its control overhead. Specifically, the controller directly communicates with compute nodes by RPC, and each compute node only receives the required messages. NG reduces the control overhead by 50.1% compared with RPC by adopting the MQ model but still results in a higher control overhead compared with VITA-based solutions. The reason is the nodes are grouped based on the physical network topology, resulting in significant differences in required messages of nodes in the same group. As for three VITA-based solutions, both VITA-SM and VITA-KM can reduce the control overhead by about 47% compared with VITA-SA. That is because VITA-SA may send the same message to multiple queues, while VITA-SM and VITA-KM only send each message to exactly one queue.

Fig. 4 shows the MQ overhead of NG and three VITA-based algorithms by changing the number of VPCs. Note that we do not evaluate this metric for RPC since RPC does not use the MQ model. The results of the MQ overhead are of a similar trend with those of the control overhead for these algorithms. That is because both control overhead and



(a) Control Overhead vs. MQ Type (b) MQ Overhead vs. MQ Type (c) Average Delay vs. MQ Type (d) Throughput vs. MQ Type

Fig. 7: The performance of proposed algorithms using different MQs.

MQ overhead are positively correlated with the total traffic amount of southbound messages. For instance, when there are 5×10^4 VPCs, the MQ overheads of VITA-SM, VITA-KM, and VITA-SA are 2.5Gbps, 2.5Gbps, and 4.9Gbps, respectively, while that of RPC is 12.4Gbps. That is, both VITA-SM and VITA-KM can reduce the MQ overhead by about 79.8% and 60.5% compared with NG and VITA-SA, respectively.

Figs. 5-6 show that the total/maximum traffic amount on compute nodes increases for all solutions with the increasing number of VPCs. RPC and NG achieve the lowest and highest total/maximum traffic amount on compute nodes among all solutions, respectively. That is because RPC using the direct communication method will not cause message redundancy, while NG using a physical host-based grouping scheme will result in high redundancy. Note that, since RPC will cause an unacceptable control overhead as shown in Fig. 3, it is not feasible in large-scale clouds. We use the total/maximum traffic amount on compute nodes of RPC as the low bound to compare with other solutions. For example, given 6×10^4 VPCs in the cloud, the total traffic amount on compute nodes is 61Gbps, 65Gbps, and 90Gbps for VITA-SA, VITA-SM, and VITA-KM, respectively, while that of NG is 198Gbps. These results mean that VITA-SM reduces the total traffic amount on compute nodes by 29% and 66% compared with VITA-KM and NG, respectively, while slightly increases the traffic amount on compute nodes by 6% compared with VITA-SA. The total/max traffic amount on compute nodes of VITA-SA is lower than that of VITA-SM because it sends messages to more queues with higher control overhead to achieve lower message redundancy.

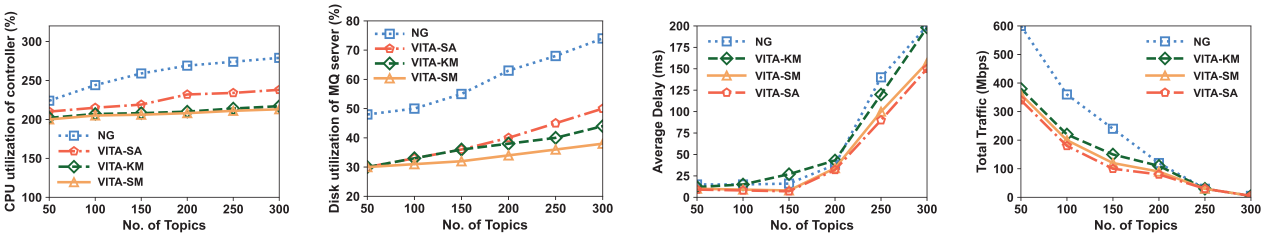
From these simulation results, we can draw some conclusions. First, as shown in Fig. 3, RPC is not feasible in large-scale clouds because it will cause unacceptable control overhead. Second, as shown in Figs. 3-6, VITA-based solutions can achieve superior performance, including lower control/MQ overhead and lower total/max traffic amount compared with NG. Third, VITA-SM reduces the total/maximum traffic amount by 29%/37% and achieves similar control/MQ overhead performance compared with VITA-KM. Fourth, compared with VITA-SA, VITA-SM reduces the control/MQ overhead by 47%/49% and increases the total/maximum traffic amount by 6%/15%.

C. System Implementation

1) Implementation on the Platform: In general, we use 10 servers running Ubuntu 18.04 with Linux kernel 5.4 to build the testbed. All the servers are equipped with a 22-core Intel Xeon 6152 processor, 128GB memory and an Intel X710 10GbE NIC. Among them, two servers are used as the controller and the message queue server, respectively. We take a small cloud deployed in GoDaddy [42] as a reference, which contains 350 compute nodes. We rely on the virtualization technology for system implementation to expand the testing topology and collect testing data conveniently. Specifically, we deploy 350 VMs, each equipped with 1 vCPU and 1GB memory, as compute nodes on the remaining 8 servers. The number of VPCs and topics is by default set to 300 and 100.

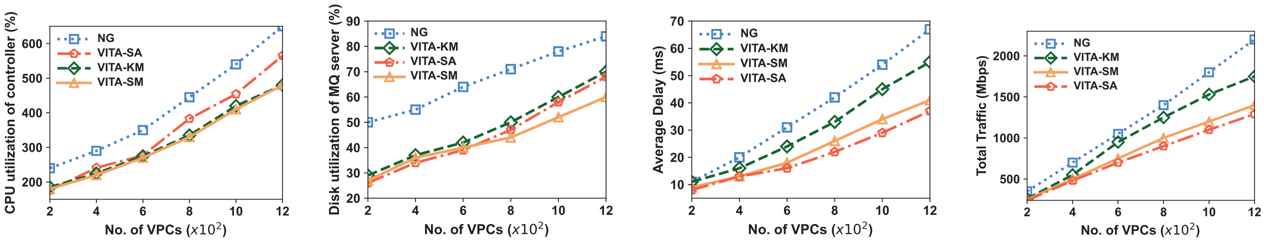
We run three sets of experiments on the platform. The expected traffic intensity for messages of each VPC is set to 1Mbps and the bandwidth constraint of each compute node is 1Gbps by default. The message-size distribution is the same as in simulations where 20% of all messages account for 80% of traffic amount. These messages are distributed in size from 512Bytes to 4MB. According to [43], we generate two types of messages: (1) unicast messages, whose sources and destinations are randomly picked, *e.g.*, IP address segment configuration messages; (2) multicast messages, which simulate the traffic with multiple destinations, *e.g.*, subnet and security group configuration messages. Each type of message accounts for half of the total traffic amount.

2) Test Results: The first set of experiments compares the overall performance of all benchmarks using three well-known MQ frameworks. Specifically, we take three open-source MQ frameworks for comparison: Apache Kafka (version 2.6.0) [44], RabbitMQ (version 3.8.19) [45], and Apache Pulsar (version 2.6.1) [46]. Kafka is the most widely deployed open-source MQ framework, and RabbitMQ is used in OpenStack. As for Pulsar, it is one of the fastest-growing MQ frameworks in recent years. The physical parameter settings of these MQ frameworks are the same as in [20]. We set 100 topics for each MQ framework and generate 200 VPCs by default. As shown in Fig. 7, VITA-SM performs better compared with NG and VITA-KM in all three MQ frameworks. Moreover, VITA-SM achieves lower control/MQ overhead, but results in higher message delay and higher total traffic amount on compute nodes than VITA-SA. That means, VITA-SM is more suitable for scenarios with limited processing capacity on the control plane or the MQ server, while VITA-SA is more suitable for scenarios with limited processing capacity on compute nodes. Note that, as shown in Figs. 7(c)-7(d), RabbitMQ achieves the lowest total traffic amount, while achieves the smallest message delivery delay, compared with the other two frameworks. The reason is that RabbitMQ aims to obtain low message transmission delay, while the total throughput cannot be guaranteed. To save the space, we only conduct a detailed



(a) Control Overhead vs. No. of Topics (b) MQ Overhead vs. No. of Topics (c) Average Delay vs. No. of Topics (d) Total Traffic vs. No. of Topics

Fig. 8: Control Overhead, MQ Overhead, Average Message Delay and Total Traffic vs. Number of Topics



(a) Control Overhead vs. No. of VPCs (b) MQ Overhead vs. No. of VPCs (c) Average Delay vs. No. of VPCs (d) Total Traffic vs. No. of VPCs

Fig. 9: Control Overhead, MQ Overhead, Average Message Delay and Total Trafiic vs. Number of VPCs

performance comparison of all solutions when using Kafka in the following since it is the most widely used framework.

The second set of experiments observes the control/MQ overhead, average message delay, and total traffic amount of NG, VITA-SA, VITA-KM and VITA-SM by changing the number of available topics in the MQ server. The results are shown in Fig. 8, where the horizontal axis is the number of topics in the MQ server, ranging from 50 to 300. No matter how many topics there are in the MQ server, NG always achieves the worst performance compared with other solutions. For example, as shown in Fig. 8(b), given 200 topics, the average disk I/O utilization of VITA-SM, VITA-KM, VITA-SA and NG is 34%, 38%, 40% and 63%, respectively. That is, VITA-SM can reduce the average disk I/O utilization by about 10.5%, 15% and 46% compared with VITA-KM, VITA-SA and NG, respectively. We should note that, as shown in Fig. 8(c), when the number of topics exceeds 200, the average message delay will increase significantly as the number of topics increases. That means the MQ server can only support a limited number of topics. Thus, we should carry out message aggregation with a proper granularity.

The third set of experiments compares the control/MQ overhead, average message delay, and total traffic amount of NG, VITA-SA, VITA-KM and VITA-SM by changing the number of VPCs in the cloud. The results are shown in Fig. 9, where the number of VPCs ranges from 200 to 1,200. As the number of VPCs increases, all performance metrics (*e.g.*, control overhead, MQ overhead, message delay and total traffic amount) increase for all algorithms. NG always achieves the worst performance compared with the other three VITA-based solutions. For example, when the number of VPCs reaches 1000, the average message delay of NG, VITA-SA, VITA-KM and VITA-SM is 54ms, 29ms, 45ms and 34ms. That means, VITA-SM can reduce the average message delay by 37% and 24.4% compared with NG and VITA-KM, respectively. VITA-based solutions are more efficient compared with NG, since southbound messages usually have VPC attributes, and VITA-based solutions aggregate messages with VPC as the granularity.

From these experimental results, we can draw some conclusions. First, as shown in Fig. 7, VITA-SM performs better in all three MQ frameworks compared with NG and VITA-KM, and achieves similar performance compared with VITA-SA. Second, Fig. 8 illustrates that the MQ server can only support a limited number of topics. Thus, we have to aggregate messages with a proper granularity. Third, the performance of NG lags behind all three VITA-based solutions for all metrics (*e.g.*, control/MQ overhead, message delay and traffic amount on compute nodes). Fourth, our proposed VITA-SM performs better than VITA-KM, especially in the metrics of message delay and total traffic amount, which shows efficiency of our proposed message aggregation algorithm. Fifth, our proposed VITA-SM and VITA-SA algorithms have different application scenarios. If the control/MQ overhead become the network bottleneck, VITA-SM is a better choice compared with VITA-SA. Conversely, if resources on compute nodes are the network bottleneck, VITA-SA is a better choice compared with VITA-SM.

V. Conclusion

In this paper, we give the system overview of VITA and formulate the VSMD problem for minimizing the total amount of messages received by compute nodes. We propose a submodular-based algorithm for this problem and analyze its approximation performance. We further consider how to extend this scheme for more scenarios. Both the simulation and experimental results show high efficiency of our proposed VITA system.

Acknowledgement

The corresponding authors of this paper are Gongming Zhao and Hongli Xu. We would also like to thank Xuwei Yang, Eric Li, James Chung for their contributions to this work. This article was supported in part by the National Science Foundation of China (NSFC) under Grants 61822210 and 62102392, and in part by the National Science Foundation of Jiangsu Province under Grant BK20210121.

REFERENCES

[1] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[2] R. Birke, A. Podzimek, L. Y. Chen, and E. Smirni, "State-of-the-practice in data center virtualization: Toward a better understanding of vm usage," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2013, pp. 1–12.

[3] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *2010 Proceedings IEEE INFOCOM*. Ieee, 2010, pp. 1–9.

[4] T. Bell, B. Bompastor, S. Bukowiec, J. C. Leon, M. Denis, J. van Eldik, M. F. Lobo, L. F. Alvarez, D. F. Rodriguez, A. Marino *et al.*, "Scaling the cern openstack cloud," in *Journal of Physics: Conference Series*, vol. 664, no. 2. IOP Publishing, 2015, p. 022003.

[5] H. Qu, O. Mashayekhi, C. Shah, and P. Levis, "Decoupling the control plane from program control flow for flexibility and performance in cloud computing," in *Proceedings of the thirteenth euays conference*, 2018, pp. 1–13.

[6] K. Zheng, L. Wang, B. Yang, Y. Sun, and S. Uhlig, "Lazyctrl: A scalable hybrid network control plane design for cloud data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 115–127, 2016.

[7] S. Maheshwari, P. Netalkar, and D. Raychaudhuri, "Disco: Distributed control plane architecture for resource sharing in heterogeneous mobile edge cloud scenarios," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 519–529.

[8] Y. Gong, B. He, and J. Zhong, "Network performance aware mpi collective communication operations in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 11, pp. 3079–3089, 2013.

[9] R. Thurlow, "Rpc: Remote procedure call protocol specification version 2," RFC 5531, May, Tech. Rep., 2009.

[10] P. Stuedi, A. Trivedi, B. Metzler, and J. Pfefferle, "Darpc: Data center rpc," in *Proceedings of the ACM Symposium on Cloud Computing*, 2014, pp. 1–13.

[11] H. Kraft and R. Johansson, "Evaluating rpc for cloud-native 5g mobile network applications," 2020.

[12] grpc. Accessed: July. 20, 2021. [Online]. Available: https://grpc.io/

[13] Apache thrift. Accessed: July. 20, 2021. [Online]. Available: https://thrift.apache.org/

[14] M. Fazio, A. Celesti, A. Puliafito, and M. Villari, "A message oriented middleware for cloud computing to improve efficiency in risk management systems," *Scalable Computing: Practice and Experience*, vol. 14, no. 4, pp. 201–213, 2013.

[15] C. Wang, H. Ni, and L. Liu, "An enhanced message distribution mechanism for northbound interfaces in the sdn environment," *Applied Sciences*, vol. 11, no. 10, p. 4346, 2021.

[16] W. Lu, J. Jackson, and R. Barga, "Azureblast: a case study of developing science applications on the cloud," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 413–420.

[17] N. M. Abd Elazim, M. A. Sobh, and A. M. Bahaa-Eldin, "Software defined networking: attacks and countermeasures," in *2018 13th International Conference on Computer Engineering and Systems (ICCES)*. IEEE, 2018, pp. 555–567.

[18] A. Hakiri, P. Berthou, A. Gokhale, and S. Abdellatif, "Publish/subscribe-enabled software defined networking for efficient and scalable iot communications," *IEEE communications magazine*, vol. 53, no. 9, pp. 48–54, 2015.

[19] J. Chen and B. Dezfouli, "Modeling control traffic in software-defined networks," in *7th IEEE International Conference on Network Softwarization (NefSoft)*, 2021.

[20] P. Dobbelaere and K. S. Esmaili, "Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *Proceedings of the 11th ACM international conference on distributed and event-based systems*, 2017, pp. 227–238.

[21] T. Rosado and J. Bernardino, "An overview of openstack architecture," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, 2014, pp. 366–367.

[22] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Performance of multi-tenant virtual networks in openstack-based cloud infrastructures," in *2014 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2014, pp. 81–85.

[23] B. Beach, S. Armentrout, R. Bozo, and E. Tsouris, "Virtual private cloud," in *Pro Powershell for Amazon Web Services*. Springer, 2019, pp. 85–115.

[24] A. Gupta, A. Mehta, L. Daver, and P. Banga, "Implementation of storage in virtual private cloud using simple storage service on aws," in *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*. IEEE, 2020, pp. 213–217.

[25] D. C. Wa, "Security in the virtual private cloud," in *Security in the Private Cloud*. CRC Press, 2016, pp. 165–176.

[26] W.-H. Liao and S.-C. Su, "A dynamic vpn architecture for private cloud computing," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*. IEEE, 2011, pp. 409–414.

[27] A. Z. Bhat, D. K. Al Shuaibi, and A. V. Singh, "Virtual private network as a service—a need for discrete cloud architecture," in *2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, 2016, pp. 526–532.

[28] L. Malina, G. Srivastava, P. Dzurenda, J. Hajny, and R. Fujdiak, "A secure publish/subscribe protocol for internet of things," in *Proceedings of the 14th international conference on availability, reliability and security*, 2019, pp. 1–10.

[29] K. An, S. Pradhan, F. Caglar, and A. Gokhale, "A publish/subscribe middleware for dependable and real-time resource monitoring in the cloud," in *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management*, 2012, pp. 1–6.

[30] L. Nie, D. Jiang, and Z. Lv, "Modeling network traffic for traffic matrix estimation and anomaly detection based on bayesian network in cloud computing networks," *Annals of Telecommunications*, vol. 72, no. 5, pp. 297–305, 2017.

[31] R. A. Memon, S. Qazi, and B. M. Khan, "Design and implementation of a robust convolutional neural network-based traffic matrix estimator for cloud networks," *Wireless Communications and Mobile Computing*, vol. 2021, 2021.

[32] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[33] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[34] A. Krause and D. Golovin, "Submodular function maximization." *Tractability*, vol. 3, pp. 71–104, 2014.

[35] H. Xu, Z. Yu, X.-Y. Li, L. Huang, C. Qian, and T. Jung, "Joint route selection and update scheduling for low-latency update in sdns," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 3073–3087, 2017.

[36] R. Gandhi, H. H. Liu, Y. C. Hu, G. Lu, J. Padhye, L. Yuan, and M. Zhang, "Duet: Cloud scale load balancing with hardware and software," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 27–38, 2014.

[37] M. Dalton, D. Schultz, J. Adriaens, A. Arefin, A. Gupta, B. Fahs, D. Rubinstein, E. C. Zermeno, E. Rubow, J. A. Docauer *et al.*, "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 373–387.

[38] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[39] A. G. Nikolaev and S. H. Jacobson, "Simulated annealing," in *Handbook of metaheuristics*. Springer, 2010, pp. 1–39.

[40] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[41] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, 2009, pp. 202–208.

[42] Godaddy. Accessed: July. 20, 2021. [Online]. Available: https://www.godaddy.com/

[43] S. Paul, R. Jain, M. Samaka, and J. Pan, "Application delivery in multi-cloud environments using software defined networking," *Computer Networks*, vol. 68, pp. 166–186, 2014.

[44] Apache kafka. Accessed: July. 20, 2021. [Online]. Available: https://kafka.apache.org/

[45] Rabbitmq. Accessed: July. 20, 2021. [Online]. Available: https://www.rabbitmq.com/

[46] Apache pulsar. Accessed: July. 20, 2021. [Online]. Available: https://pulsar.apache.org/