Southbound Message Delivery With Virtual Network Topology Awareness in Clouds

Gongming Zhao, *Member, IEEE*, Luyao Luo, *Graduate Student Member, IEEE*, Hongli Xu, *Member, IEEE*, Chun-Jen Chung, and Liguang Xie, *Senior Member, IEEE*

***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 [2]. 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 [3], [4]. The control plane processes tenants' requests, and sends network configuration messages, also called *southbound messages*, to computes nodes [5]. 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 [6]. 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 [7]–[10].

A natural method to deliver southbound messages is direct end-to-end transmission via message passing interfaces (MPI) [11] or remote procedure call (RPC) [12]. 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 [13]. This insight has been discovered by the experiments [14], in which gRPC [15] and Apache Thrift [16], 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 [17], [18]. 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 [19], [20]. 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 [21], [22]. 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.

Manuscript received 25 February 2022; revised 31 May 2022; accepted 3 July 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. Wu. This work was supported in part by the National Science Foundation of China (NSFC) under Grant 62102392, Grant 62132019, and Grant 61936015; and in part by the National Science Foundation of Jiangsu Province under Grant BK20210121. Some preliminary results of this paper were published in the Proceedings of IEEE INFOCOM 2022 DOI: 10.1109/INFOCOM48880.2022.9796938. *(Corresponding author: Hongli Xu.)*

Gongming Zhao, Luyao Luo, and Hongli Xu are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou, Jiangsu 215123, China (e-mail: gmzhao@ustc.edu.cn; lly00@mail.ustc.edu.cn; xuhongli@ustc.edu.cn).

Chun-Jen Chung and Liguang Xie are with Futurewei Technologies Inc., Bellevue, WA 98004 USA (e-mail: cchung@futurewei.com; lxie@futurewei.com).

Digital Object Identifier 10.1109/TNET.2022.3190730

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 [6]), 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 [23] 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 [24]. That is, the compute nodes are 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 [25]. Its implementation by cloud provider is Virtual Private Cloud (VPC) [26], [27], 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 [26]–[28]. 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.

The rest of this paper is organized as follows. Section II gives background and motivation through an example. Section III presents the system overview of VITA. Section IV formulates the southbound message delivery problem and proposes two message aggregation and distribution algorithms for different message delivery scenarios. Section V gives the subscription procedure of the VITA agent on each compute node. The results of the testbed and large-scale simulations are presented in Section VI. Section VII gives some related works for this paper and Section VIII concludes the paper.

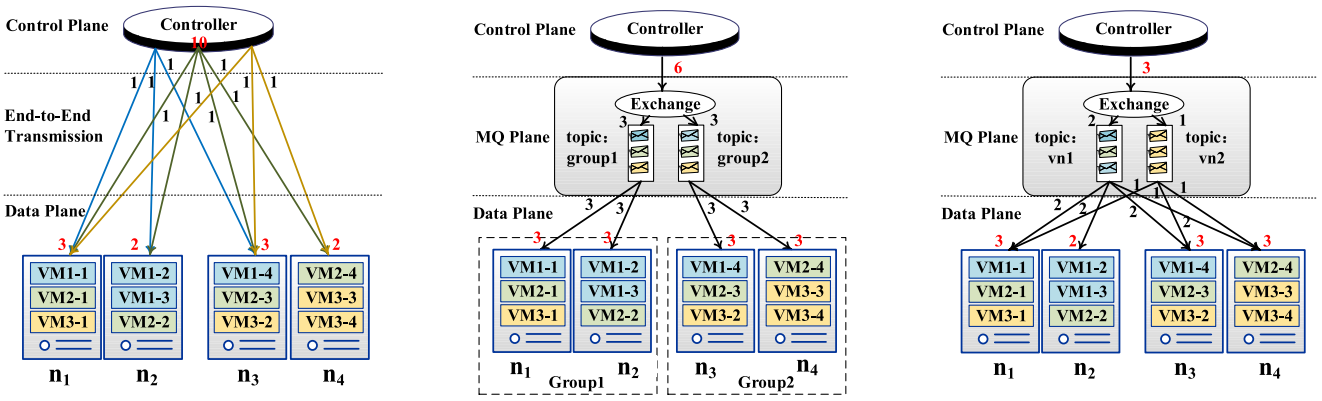
II. Background and Motivation

In this section, we first give an introduction to southbound message delivery in clouds. Then we provide an example to illustrate the pros and cons of both RPC and NG, which motivate the idea of virtual network topology-aware southbound message delivery scheme.

A. Southbound Message Delivery

Southbound message, also called network configuration message, is a vital information carrier for the interaction between the control plane and the data plane in the cloud [5], [7]. As the size of the cloud network increases, so does the number of southbound messages. Thus, how to achieve resource-saving and efficient southbound message delivery has become a fundamental problem in clouds [8]–[10].

There are two typical methods for southbound message delivery in cloud networks. A natural one is the distributed end-to-end communication model [29], which provides direct communications between the controller and compute nodes. For example, as one of the most common protocols in distributed frameworks, Remote Procedure Call (RPC) [12], [13] transmits serialized messages via custom TCP protocols or HTTP. As cloud network scale increases, the distributed communications result in massive control overhead, which leads to a significant increase in message delivery delay and a drop in throughput. To cope with this problem, Message Queue (MQ) [23] is adopted by many cloud platforms (*e.g.*, RabbitMQ in OpenStack [30] and Amazon Message Queuing Service in AWS [27]). MQ, as an independent component in distributed systems, is a more scalable southbound message delivery solution, which provides decoupling and asynchronous communication of the control plane and data plane. Communications between the control /data plane and the MQ server also relies on the TCP protocol. Considering a large amount of compute nodes and a relatively small number of queues supported by an MQ server, the Node Grouping (NG) [24] method divides those nodes into several groups according to some underlay network attributes *e.g.*, subnet. Then the controller sends the messages of the compute nodes in the same group to the same queue. The compute nodes



(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).

themselves receive messages in one or more queues by their own needs. However, the NG method inevitably brings redundancy to southbound messages.

As the concept of Virtual Private Cloud (VPC) [26], [31], [32] emerges in both the public and private clouds, the message itself usually contains the attributes of VPC. VPC builds an isolated virtual network environment for cloud servers, cloud containers, cloud databases, and other resources that users configure and manage independently, improves the security of users' cloud resources, and simplifies users' network deployment. With the help of VPC, we can study a new perspective on the problem of southbound messaging through the virtual network topology.

B. A Motivation Example

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 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.

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

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

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.

C. 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 [14]. 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 [23]. 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 [31], [33]. 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.

III. VITA Overview

A. System Overview

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. 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 IV.

VMs belonging to different VPCs are distributed on different compute nodes in the data plane. For more efficient implementation, a control agent is designed on each node to be responsible for subscribing to topics, distinguishing messages,

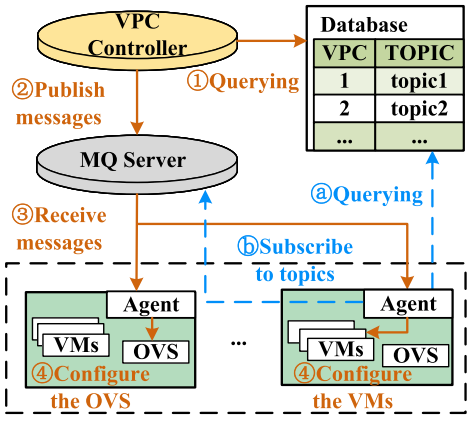


Fig. 2. System overview of VITA. VITA mainly consists of three parts. The control plane is responsible for determining the correspondence between VPCs and topics. The data plane is responsible for subscribing to topics and configuring VMs or OVS. The message queue server is responsible for the asynchronous communication between the control and data planes. Besides, a database maintains a mapping table from VPCs to topics.

and parsing requests. The agent manages all virtual machines on the node and knows which VPCs they belong to. We give a detailed design for the agent in Section V to meet the dynamic update requirement.

As an essential component, the MQ server is responsible for the asynchronous communication between the control and data planes. However, this is not the focus of our paper. In Section VI, we test the performance of our proposed algorithms on different open-source MQs to illustrate the efficiency and versatility of the VITA system.

B. Workflow of VITA

Fig. 2 also briefly describes the system workflow. The system process is mainly triggered by two events. 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 corresponding queues in the MQ server via the specified TCP port and asynchronously waits for the reply of the processing result. The other 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. 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 or OVS (Open vSwitch) according to the content of the message and return the operation result to the control plane.

With the help of MQ and the database shown in Fig. 2, VITA can realize the decoupling of the control plane and the data plane. That is, the control plane and data plane avoid

TABLE II
IMPORTANT NOTATIONS

Notations	Semantics
N	the set of compute nodes
V	the set of VPCs
T	the set of topics
$f(v)$	expected traffic intensity of VPC v
$s(n)$	message processing capability of node n
$b(n)$	traffic amount of node n
y_v^t	whether the controller will publish messages of VPC v to topic t or not
z_n^t	whether the compute node n will subscribe to topic t or not
Γ_n^v	whether the compute node n contains VMs of VPC v or not
$R(V)$	total traffic amount of messages of VPCs in V
$Sub(V)$	the set of compute nodes which contain VMs of any VPC in V
Φ	the set of disjoint subsets of V
$\varphi(\Phi)$	traffic amount reduction achieved by dividing VPCs according to Φ

directly communications, and changes in the data plane will not directly affect the control plane. The control plane needs to know not the physical location (*e.g.*, IP address) of the node, but the message identity in the virtual network (*e.g.*, VPC ID). Due to the synchronous communication service provides by the message queue server, the communication between the control plane and the message queue is asynchronous, and we use a call-back function to handle message processing results. In this way, the control plane does not have to continuously wait for the processing result when concurrent messages are sent. The communication between compute nodes and the message queue is also asynchronous, which indicates that the processing of the message on the compute node does not hinder the delivery of the messages. The operation in the data plane will not block the control plane, and will not significantly affect the system throughput when dealing with burst southbound traffic.

IV. VITA CONTROL PLANE DESIGN

Determining the correspondence between VPCs and topics is the key step in the VITA control plane. To achieve efficient southbound message delivery with low message redundancy, we first formulate the virtual network southbound message delivery (VSMD) problem in our design. Then we present an efficient approximation algorithm based on a submodular function and analyze its approximation performance. Furthermore, we extend this scheme to more practical scenarios.

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 $N = \{n_1, n_2, \ldots, n_{|N|}\}$ to represent the set of compute nodes. The set of VPCs in the cloud is denoted as V=$\{v_1, v_2, \ldots, v_{|V|}\}$. 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 [34], [35]. 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 $T = \{t_1, t_2, \ldots, t_K\}$, 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 [36], [37], 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:

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

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. Specifically, $z_n^t \cdot y_v^t$ represents whether compute node n receives messages of VPC v through the queue specified as topic t or not, and $\sum_{t\in T} z_n^t y_v^t$ means whether compute node n can receive messages of VPC v or not. When $\Gamma_n^v = 1$, compute node n must receive messages of VPC v. 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.

Proof: Our VSMD problem remains NP-hard even if the mappings between VPCs and topics are determined (*i.e.*, variables y_v^t are fixed). Under this case, the problem turns to be a Weighted Set Covering Problem (WSCP) [38] for each compute node. More specifically, each node tends to receive all the required messages with as few redundant messages as possible by selecting several weighted sets (*i.e.*, sets of messages in different queues). Since the Weighted Set Covering Problem is a special case of our problem, we can conclude that the VSMD problem is NP-hard. □

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 $\{V_1, V_2, \ldots, V_K\}$, 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 in Eq. (3), where $\sum_{i=1}^{K} R(V_i) = \sum_{n\in N} b(n)$. Then the optimal solution to Eq. (3) is also the optimal solution to Eq. (1).

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

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:

$$R(V-M)-R(V-M-A) \leq R(V-M') - R(V-M'-A) \quad (7)$$

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

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

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,

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

We call the sets satisfying Eq. (9) as *feasible sets*. The feasible sets can be explored efficiently by simply performing a depth-first search [39] on the VPC set V. During each iteration of the depth-first search, we gradually expand the candidate feasible set by adding untraversed VPC and simultaneously update the traffic of the affected compute nodes.

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 [40].

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 $A \cap S = \emptyset, \forall S \in \Phi$, 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). For the submodular function, this result is the best that can be achieved with any efficient algorithm. In fact, [41] proved that any algorithm that is allowed to only evaluate the submodular function at a polynomial number of sets will not be able to obtain an approximation guarantee better than $(1-1/e)$. □

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

1: **Step 1: Initialization**
2: Compute the set of feasible sets Π
3: $\Phi \leftarrow \emptyset$
4: **Step 2: Greedy Selection**
5: **while** $|\Phi| \leq K-1$ **do**
6: Set $tmp \leftarrow 0, opt \leftarrow 0$
7: **for** $S \in \Pi - \Phi$ **do**
8: $tmp \leftarrow \varphi(\Phi \cup \{S\})$
9: **if** $tmp > opt$ **then**
10: $opt \leftarrow tmp$, $S^* \leftarrow S$
11: **end if**
12: **end for**
13: $\Phi \leftarrow \Phi + \{S^*\}$
14: **end while**
15: $\Phi \leftarrow \Phi + \{V - \bigcup_{S \in \Phi} S\}$
16: **Step 3: Assignment of VPCs and Topics**
17: $i \leftarrow 1$
18: **for** $S \in \Phi$ **do**
19: Set $y_v^{t_i} = 1$ if $v \in S$
20: $i \leftarrow i+1$
21: **end for**

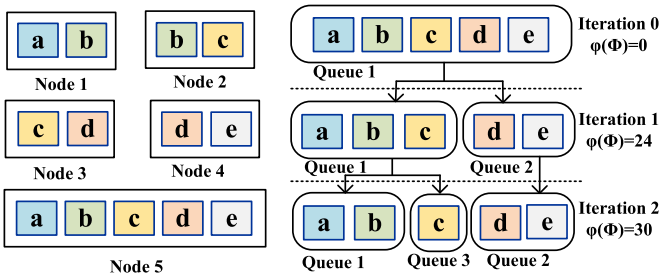


Fig. 3. An example of SM-SMD algorithm. There are 5 nodes, 5 VPCs, 3 queues and 15 messages to deliver (1, 2, 3, 4, and 5 messages are sent to VPCs a, b, c, d, and e). Left: VPC distribution in 5 nodes. Right: details of 2 iterations.

Example. We give a simple example to demonstrate the flow of the algorithm and how it finds the optimal solution, as shown in Fig. 3. In this example, we assume that there are 5 VPCs, 3 queues and 15 messages to deliver (1, 2, 3, 4, and 5 messages are sent to VPCs a, b, c, d, and e, respectively). The detailed steps of the algorithm are shown in Algorithm 1. At the first iteration, the maximum value of $\varphi(\Phi)$ we can get is 24 by dividing VPCs a, b, and c into one set (queue 1), and VPCs d and e into another set (queue 2). Then in the second iteration, we get the maximum value of $\varphi(\Phi)$ by dividing VPCs a and b into one set (queue 1), and VPC c into another set (queue 3). As a result, we obtain $R(V) - \sum_{i=1}^{K} R(V_i) = 30$, which is also the optimal solution to Eq. (3).

We should note that the number of feasible sets may be exponential. However, the work [42] 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 [4], [43]. 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 [44] 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 [44], [45]. We determine the parameters based on the work [44] to achieve a high probability for converging to the global optimal solution.

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 $z_n^t = \Gamma_n^v y_v^t$
3: Init temperature $\mathcal{T} = t_0$, $k = 0$
4: **while** $\mathcal{T} \geq t_m$ **do**
5: **while** $k \leq L$ **do**
6: Select a random VPC v and a random topic t
7: Set $y_v^t \leftarrow 1 - y_v^t$.
8: Set Δ to be difference of total traffic amount by topic re-selection.
9: **if** $\Delta > 0$ **then**
10: Set $y_v^t \leftarrow 1 - y_v^t$ with probability $1 - e^{-\frac{\Delta}{\mathcal{T}}}$
11: **end if**
12: $k \leftarrow k + 1$
13: **end while**
14: Set $k = 0$
15: $\mathcal{T} \leftarrow \alpha\mathcal{T}$
16: **end while**
17: Output the results

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.

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 $O(L \cdot \log_\alpha(t_m/t_0) \cdot |N|)$. Since the simulated annealing algorithm utilizes the result of the submodular algorithm as input, it can obtain a relatively good initial solution at the beginning, which greatly increases the probability of finding the optimal solution. At the same time, although we have no guarantee of the sub-optimal solution accuracy, we can guarantee that the solution obtained by SA-SMD can reduce more message redundancy than the sub-optimal result of the submodular algorithm SM-SMD.

V. VITA Agent Design

Although the VITA controller determines the mapping relationship between VPCs and topics, there are still two issues to consider when implementing topic subscription for each compute node. The first issue is that how compute nodes subscribe or unsubscribe from relevant topics when VMs are added, removed, or migrated on them accurately and quickly. The second issue is that when the traffic in the network changes and the VPC and topic mapping needs to be updated, how does the controller notify the compute nodes to update so that messages can be transmitted in an orderly manner without loss. To address the two problems, this section describes the topic subscription of the VITA agent, which offers a stable and consistent southbound messaging service.

A. Dynamic Subscription

To realize the decoupling of control and data planes, we rely on a metadata database to store the topic mapping information as shown in Fig. 2, which enables the agent to quickly establish a connection with the MQ server when the node initializes or recovers from a crash. When a VM belonging to a new VPC is added to a compute node, the agent should update the topics subscribed by this node. To achieve the dynamic subscription, the agent first gets the VPC to which the VM belongs. If it has not subscribed to the related topic, the agent will query the database and get the corresponding topic. Then it subscribes to this topic and begins to receive messages from the corresponding queue. The same operation will also be

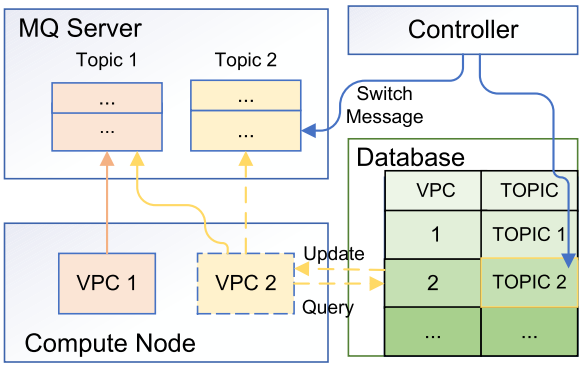


Fig. 4. Two dynamic subscription procedures of VITA agent. 1) The VM belonging to VPC 2 is newly added. Then the agent queries the database and subscribes to the corresponding Topic 2 (dotted yellow line). 2) The mapping relationship of VPC 2 is switched from Topic 2 to Topic 1. Then the controller sends a switch message to Topic 2 and modifies the database (solid blue line). When the agent receives the switch message, it will unsubscribe from Topic 2 and subscribe to Topic 1 (solid yellow line).

performed when a VM is migrated or updated. With the help of the database and MQ server, the control plane and the data plane are completely decoupled, and the compute node does not need to frequently send requests to the controller.

We illustrate this dynamic subscription procedure more intuitively with an example, as shown in Fig. 4. There are two VPCs (VPC 1 and 2) in Fig. 4 and two related topics (Topic 1 and 2). The compute node contains two VMs, one of which is a newly added VM belonging to VPC 2. To obtain the topic corresponding to VPC 2, the VITA agent queries the database and gets Topic 2 as a result. Since the agent has not subscribed to Topic 2 yet, it subscribes to this topic and receives messages from the corresponding queue (represented by dotted yellow lines). This way, compute nodes can implement dynamic subscriptions without communicating with the controller.

B. Seamless Switching

Considering the dynamics of user requests and network environment, the VITA controller will periodically run the algorithms in Section IV to update the subscription relationship between VPCs and topics to ensure the efficiency of the VITA system. However, this operation may result in high message delay or message loss. If the control plane cannot notify the data plane to switch topics in time when the mapping relationship between VPCs and topics changes, the message delivery delay will increase significantly due to the decoupling of the control plane and the data plane. Moreover, message loss occurs when a node does not wholly receive the messages stored in the subscribed queue.

To solve these two problems, we design a new message type called `switch message`. The `switch message` includes the previous and next topics for one VPC. When the VITA controller updates the mapping relationship between a VPC and a topic, it will first modify the database. Then the controller constructs a `switch message` and sends it to the MQ server with the previous topic. On receiving the `switch message`, the messages in the previous queue have been entirely consumed. Then, the agent unsubscribes from the previous topic if it is no longer needed and subscribes to the new topic if it has not been subscribed before. Through the `switch message`, the agent can switch seamlessly and timely without message loss.

Fig. 4 also illustrates the switching procedure, where the mapping relationship of VPC 2 is switched from Topic 2 to Topic 1. The controller firstly updates the database and sends a switch message (VPC 2: from Topic 2 to Topic 1) to Topic 2 (represented by solid blue lines). When the agent receives the switch message from Topic 2, it will unsubscribe from Topic 2 and then subscribe to Topic 1 immediately (represented by the solid yellow line). This way, the agent achieves seamless topic switching and receives all the required messages.

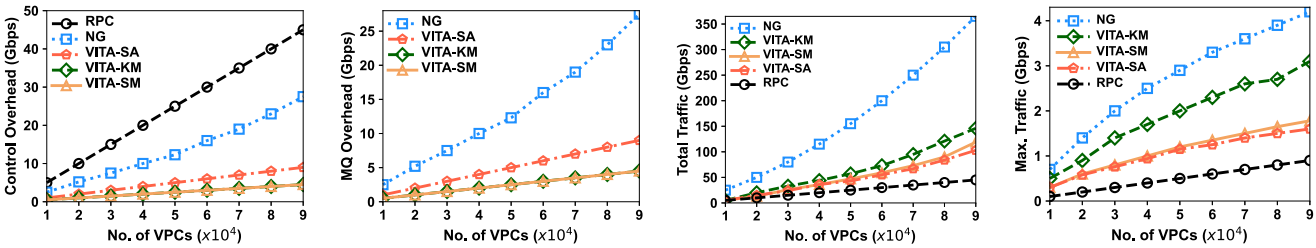
VI. Performance Evaluation

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 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 [23], 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. Specifically, considering that the performance of a simulated annealing algorithm generally depends on values of the parameters, we set our algorithm parameters as in [46], where L is 16 times the number of VPCs, $t_0 = 1000, t_m = 0.05$ and $\alpha = 0.95$. 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 [12], which is a widely used method in distributed microservice framework for



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

Fig. 5. Control overhead, MQ overhead, total and maximum traffic vs. number of VPCs.

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 [24], 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 [47], 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.

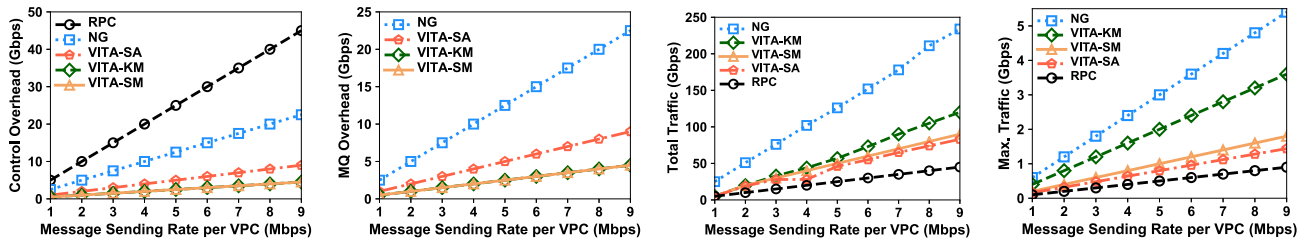
B. Simulation Evaluation

We refer to a practical private cloud deployed in CERN (European Organization for Nuclear Research) [6] 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 default 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 [48].

To analyze the performance of VITA-based algorithms in different cloud scenarios, we test the scalability performance by scaling the number of VPCs in the public cloud and by expanding the expected message traffic of each VPC in the private cloud. In the first set of simulations, 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. 5(a)-5(d). Specifically, Fig. 5(a) 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.

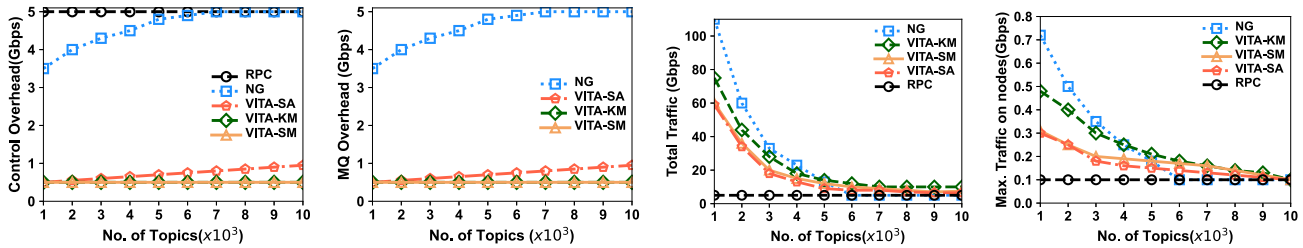
Fig. 5(b) shows the MQ overhead of NG and three VITA-based algorithms by changing the number of VPCs. Note that we do not evaluate this metric for RPC since RPC does not use the MQ model. The results of the MQ overhead are of a similar trend with those of the control overhead for these algorithms. That is because both control overhead and 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(c)-5(d) 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. 5(a), 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



(a) Control Overhead vs. Message Sending Rate (b) MQ Overhead vs. Message Sending Rate (c) Total Traffic vs. Message Sending Rate (d) Max. Traffic vs. Message Sending Rate

Fig. 6. Control overhead, MQ overhead, total and maximum traffic vs. message sending rate.



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

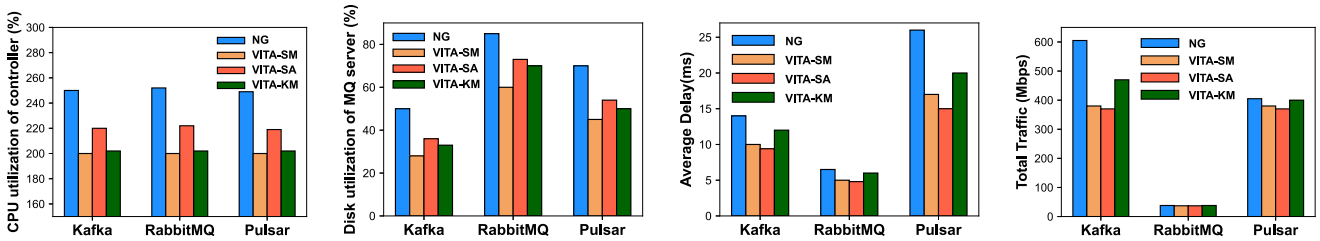
Fig. 7. Control overhead, MQ overhead, total and maximum traffic vs. number of topics.

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.

The second set of simulations shows the performance of the proposed algorithms under different expected traffic intensities. Figs. 6(a)-6(d) shows the control overhead, the MQ overhead, and the total/maximum traffic amount on compute nodes by changing the message sending rate of each VPC from 1Mbps to 9Mbps. The number of VPCs is set as 1×10^4 by default. As the message sending rate increases, all performance metrics (*i.e.*, control overhead, MQ overhead, total and maximum traffic amount) increase for all algorithms, with VITA-based solutions achieving better performance than the other solutions. For example, when the message sending rate of each VPC reaches 6Mbps, the control overhead of VITA-SM is 2.5Gbps, while the control overhead of RPC and NG is 25Gbps and 10.6Gbps, respectively. More specifically, VITA-SM reduces the control overhead by about 90% and 40% compared with RPC and NG, respectively. Among VITA-based solutions, VITA-SM works better than VITA-KM with the message sending rate increasing. Meanwhile, VITA-SA has a higher control/MQ overhead compared with VITA-SM but a lower total/max traffic amount. It means that VITA-SA trades for lower message redundancy by sacrificing a small amount of control plane resources compared with VITA-SM.

Since the number of topics greatly impacts the algorithms' performance, we compare NG, VITA-SA, VITA-KM, and VITA-SM by changing the number of available queues (topics) in the MQ server. The results are shown in Figs. 7(a)-7(d), where the horizontal axis is the number of topics in the MQ server, ranging from 1×10^3 to 10×10^3.

Figs. 7(a) and 7(b) show the control/MQ overhead with the number of topics increasing. In comparison, the proposed VITA-SM solution has the lowest control/MQ overhead. For example, given 5×10^3 topics, the control overhead of VITA-SM is 0.5Gbps while that of RPC is 5Gbps; the MQ overhead of VITA-SM is 0.5Gbps while that of NG is 4.8Gbps. More specifically, VITA-SM reduces the control/MQ overhead by about 90% compared with NG, and VITA-KM has the same control/MQ overhead as VITA-SM. That is because both VITA-SM and VITA-KM will only send each message to the corresponding topic once. The control/MQ overhead of VITA-SA is slightly higher than VITA-SM since it may send a message to multiple topics. Figs. 7(c) and 7(d) show that the total/maximum traffic on compute nodes of NG, RPC, VITA-SM, VITA-SA, and VITA-KM decreases as the number of VPCs increases. However, RPC has the minimum traffic amount on nodes since no redundant message is received. NG has the highest total/maximum traffic on nodes when topics are less than 4×10^3. For example, when there are 3×10^3 topics, the total traffic for NG, RPC, VITA-SM are 33Gbps, 28Gpbs, 20Gpbs, and the maximum traffic on each node for NG, RPC, VITA-SM are 0.35Gbps, 0.3Gbps, and 0.21Gbps, respectively. Note that when the number of topics is more than 5500, we can assign a topic for each node. Thus, each node will not receive redundant messages from other nodes, and the performance of NG will be the same as RPC.



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

Fig. 8. The performance of proposed algorithms using different MQs.

From these simulation results, we can draw some conclusions. First, as shown in Fig. 5(a), RPC is not feasible in large-scale clouds because it will cause unacceptable control overhead. Second, as shown in Figs. 5(a)-6(d), 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%. Fifth, as shown in Figs. 7(a)-7(d), VITA-based solutions outperform NG with different number of topics, especially when the number of topics is small.

C. System Implementation

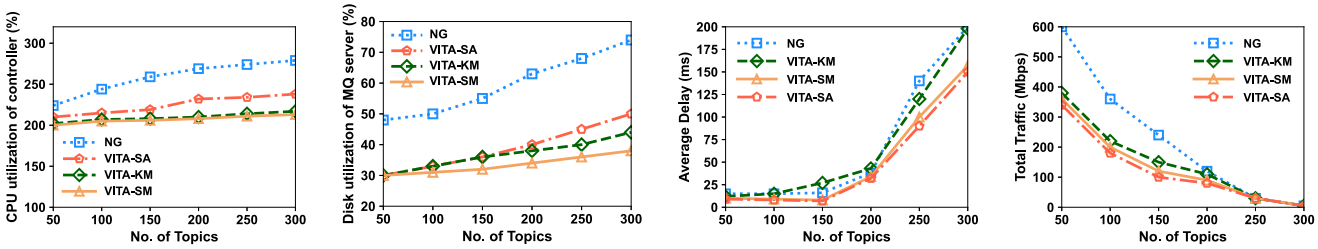
1) Implementation on the Platform: In general, we use 10 servers running Ubuntu 18.04 with Linux kernel 5.4 to build the testbed. All the servers are equipped with a 22-core Intel Xeon 6152 processor, 128GB memory and an Intel X710 10GbE NIC. Among them, two servers are used as the controller and the message queue server, respectively. We take a small cloud deployed in GoDaddy [49] 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 [50], 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) [51], RabbitMQ (version 3.8.19) [30], and Apache Pulsar (version 2.6.1) [52]. 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 [23]. We set 100 topics for each MQ framework and generate 200 VPCs by default. As shown in Fig. 8, 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. 8(a)-8(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 performance comparison of all solutions when using Kafka in the following since it is the most widely used framework.

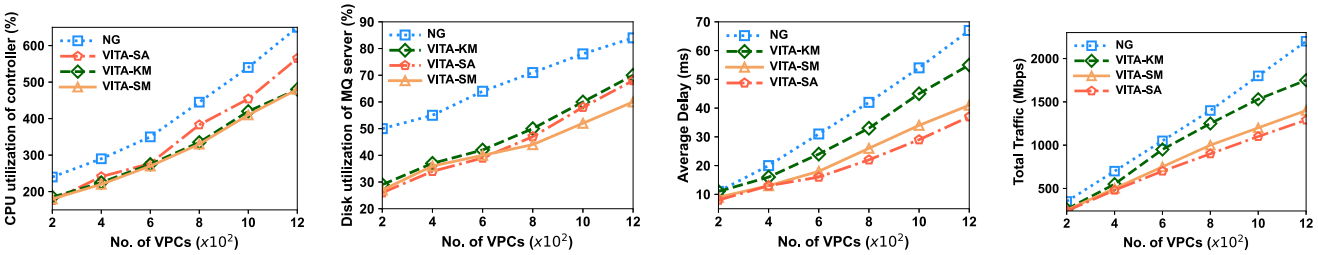
The second set of experiments observes the control/MQ overhead, average message delay, and total traffic amount of NG, VITA-SA, VITA-KM and VITA-SM by changing the number of available topics in the MQ server. The results are shown in Fig. 9, where the horizontal axis is the number of topics in the MQ 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. 9(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. 9(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



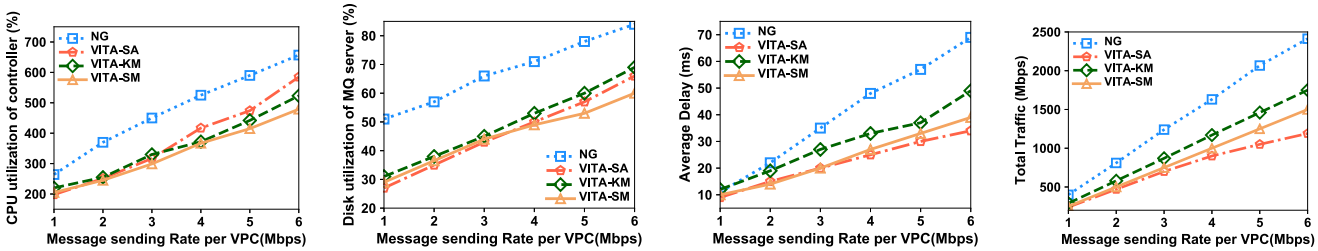
(a) Control Overhead vs. No. of Topics (b) MQ Overhead vs. No. of Topics (c) Average Delay vs. No. of Topics (d) Total Traffic vs. No. of Topics

Fig. 9. 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. 10. Control overhead, MQ overhead, average message delay and total traffic vs. number of VPCs.



(a) Control Overhead vs. Message Sending Rate (b) MQ Overhead vs. Message Sending Rate (c) Average Delay vs. Message Sending Rate (d) Total Traffic vs. Message Sending Rate

Fig. 11. Control overhead, MQ overhead, average message delay and total traffic vs. message sending rate per VPC.

number of VPCs in the cloud. The results are shown in Fig. 10, 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.

The fourth set of experiments shows the control/MQ overhead, average message delay, and total traffic amount of NG, VITA-SA, VITA-KM, and VITA-SM by changing sending rate of messages in the cloud. The results are shown in Fig. 11, where the message sending rate per VPC varies from 1Mbps to 6Mbps. As the message sending rate increases, all performance metrics (*e.g.*, control overhead, MQ overhead, message delay, and total traffic amount) increase for all algorithms, with NG always achieving the worst performance compared with the other three VITA-based solutions. For example, when the message sending rate per VPC reaches 4Mbps, the total traffic amount of NG, VITA-SA, VITA-KM, and VITA-SM is 1.64Gbps, 1.01Gbps, 1.23Gbps, and 9.2Gbps, respectively. That means VITA-SM can reduce the total traffic amount on compute nodes by about 38.4% and 17.9% compared with NG and VITA-KM, respectively. VITA-SA and VITA-SM work better than the other two algorithms at different message sending rates. Meanwhile, VITA-SM saves more resources in the control plane (*e.g.*, CPU resources of the controller), while VITA-SA cares more about the data plane (*e.g.*, bandwidth of compute nodes).

From these experimental results, we can draw some conclusions. First, as shown in Fig. 8, 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. 9 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.

VII. Related Works

In this section, we summarize recent research works on cloud computing, virtualization technologies, and prior efforts on message queue.

A. Cloud Computing

Research on cloud computing is popular in recent years. Different from classic networks, cloud computing is gaining a great scope towards IT industries, academics, and individual users because of its ease of use, on-demand access to network resources, minimal management cost [53]. Some works propose new cloud network architectures [4], [9], [54] to support fast-growing user scale and traffic demand. Some works apply VM scheduling policies [55]–[57] to achieve energy efficiency and minimize task/flow completion times. Some works propose on-demand routing protocols [58]–[60] to achieve high bandwidth and low latency in clouds. Distributed services produce significant network traffic inside clouds. To address it, some resource management frameworks [7], [61]–[63] are proposed. However, such frameworks will add more complexity and control overhead to the whole cloud network management.

B. Virtualization

Virtualization technologies partition hardware and thus provide flexible and scalable computing platforms. Virtual machine techniques, such as VMware [64] and Xen [65], offer virtualized IT-infrastructures on demand. Virtual network advances, such as VPN [32], [33] and VPC [26], [31], [32], support users with a customized network environment to access cloud resources. Virtualization techniques are the bases of cloud computing since they render flexible and scalable hardware services. Some propose new underlying technologies [4], [66], [67] to simplify cloud virtualization. Some utilize virtualization for security [53], [68]–[70] in cloud networks. Moreover, cloud computing platforms (*e.g.*, OpenStack [24],CloudStack [71], Eucalyptus [72], OpenNebula [73]), are mainly deployed in public and private clouds as an infrastructure-as-a-service (IaaS), providing virtual servers and other resources to users.

C. Message Queue

The idea of using Message Queue (MQ) in clouds or data centers has emerged since one decades ago but lacked attention until the recent rapid expansion of cloud network scale [19], [74], [75]. Eqs [76] presents an elastic message queue architecture and a scaling algorithm that can be adapted to any message queue to make it scale elastically. The authors of [77] propose a hybrid decentralized practical byzantine fault tolerance blockchain Framework with two-step verification for OpenStack message queue service. WaggleDB [78] builds a set of protocols and a cloud-based data streaming infrastructure in case that each tier can be scaled by adding more independent resources provisioned on-demand in the cloud. However, all these MQ frameworks are either not user-friendly or not paying attention to the redundancy problems caused by MQ.

VIII. 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.

References

[1] L. Luo, G. Zhao, H. Xu, L. Xie, and Y. Xiong, "VITA: Virtual network topology-aware southbound message delivery in clouds," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2022, pp. 630–639.

[2] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Gener. Comput. Syst.*, vol. 79, pp. 849–861, Feb. 2018.

[3] Y. Zhai, G. Zhao, H. Xu, Y. Zhao, J. Liu, and X. Fan, "Towards robust multi-tenant clouds through multi-constrained VM placement," in *Proc. IEEE/ACM 29th Int. Symp. Quality Service (IWQOS)*, Jun. 2021, pp. 1–6.

[4] M. Dalton *et al.*, "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization," in *Proc. 15th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2018, pp. 373–387.

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

[6] T. Bell *et al.*, "Scaling the CERN openstack cloud," *J. Phys., Conf.*, vol. 664, no. 2, 2015, Art. no. 022003.

[7] J. Barrameda and N. Samaan, "A novel statistical cost model and an algorithm for efficient application offloading to clouds," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 598–611, Jul. 2018.

[8] 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 *Proc. 13th EuroSys Conf.*, Apr. 2018, pp. 1–13.

[9] K. Zheng, L. Wang, B. Yang, Y. Sun, and S. Uhlig, "LazyCtrl: A scalable hybrid network control plane design for cloud data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 115–127, Jan. 2017.

[10] S. Maheshwari, P. Netalkar, and D. Raychaudhuri, "DISCO: Distributed control plane architecture for resource sharing in heterogeneous mobile edge cloud scenarios," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Nov. 2020, pp. 519–529.

[11] Y. Gong, B. He, and J. Zhong, "Network performance aware MPI collective communication operations in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 11, pp. 3079–3089, Nov. 2015.

[12] R. Thurlow, *RPC: Remote Procedure Call Protocol Specification Version 2*, document RFC 56531, 2009. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5531

[13] P. Stuedi, A. Trivedi, B. Metzler, and J. Pfefferle, "DaRPC: Data center RPC," in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 1–13.

[14] H. Kraft and R. Johansson, "Evaluating RPC for cloud-native 5G mobile network applications," M.S. thesis, Dept. Comput. Sci. Eng., Univ. Gothenburg, Sweden, 2020.

[15] GRPC. *GRPC: A High Performance, Open Source Universal RPC Framework*. Accessed: Jul. 20, 2021. [Online]. Available: https://grpc.io/

[16] Apache Thrift. *Apache Thrift: For Scalable Cross-Language Services Development*. Accessed: Jul. 20, 2021. [Online]. Available: https://thrift.apache.org/

[17] M. Fazio, A. Celesti, A. Puliafito, and M. Villari, "A message oriented middleware for cloud computing to improve efficiency in risk management systems," *Scalable Comput., Pract. Exper.*, vol. 14, no. 4, pp. 201–213, Jan. 2014.

[18] C. Wang, H. Ni, and L. Liu, "An enhanced message distribution mechanism for northbound interfaces in the SDN environment," *Appl. Sci.*, vol. 11, no. 10, p. 4346, May 2021.

[19] W. Lu, J. Jackson, and R. Barga, "AzureBlast: A case study of developing science applications on the cloud," in *Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput. (HPDC)*, 2010, pp. 413–420.

[20] N. M. A. Elazim, M. A. Sobh, and A. M. Bahaa-Eldin, "Software defined networking: Attacks and countermeasures," in *Proc. 13th Int. Conf. Comput. Eng. Syst. (ICCES)*, Dec. 2018, pp. 555–567.

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

[22] J. Chen, A. Gopal, and B. Dezfouli, "Modeling control traffic in software-defined networks," in *Proc. IEEE 7th Int. Conf. Netw. Softwarization (NetSoft)*, Jun. 2021, pp. 258–262.

[23] P. Dobbelaere and K. S. Esmaili, "Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *Proc. 11th ACM Int. Conf. Distrib. Event-Based Syst.*, 2017, pp. 227–238.

[24] T. Rosado and J. Bernardino, "An overview of OpenStack architecture," in *Proc. 18th Int. Database Eng., Appl. Symp. (IDEAS)*, 2014, pp. 366–367.

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

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

[27] A. Gupta, A. Mehta, L. Daver, and P. Banga, "Implementation of storage in virtual private cloud using simple storage service on AWS," in *Proc. 2nd Int. Conf. Innov. Mech. Ind. Appl. (ICIMIA)*, Mar. 2020, pp. 213–217.

[28] Q. Zhang *et al.*, "Zeta: A scalable and robust east-west communication framework in large-scale clouds," in *Proc. 19th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2022, pp. 1231–1248.

[29] K. Qureshi and H. Rashid, "A performance evaluation of RPC, Java RMI, MPI and PVM," *Malaysian J. Comput. Sci.*, vol. 18, no. 2, pp. 38–44, 2005.

[30] RabbitMQ. *RabbitMQ: The Most Widely Deployed Open Source Message Broker*. Accessed: Jul. 20, 2021. [Online]. Available: https://www.rabbitmq.com/

[31] A. Z. Bhat, D. K. A. Shuaibi, and A. V. Singh, "Virtual private network as a service—A need for discrete cloud architecture," in *Proc. 5th Int. Conf. Rel., Infocom Technol. Optim. (ICRITO)*, Sep. 2016, pp. 526–532.

[32] T. Lackorzynski, S. Kopsell, and T. Strufe, "A comparative study on virtual private networks for future industrial communication systems," in *Proc. 15th IEEE Int. Workshop Factory Commun. Syst. (WFCS)*, May 2019, pp. 1–8.

[33] W.-H. Liao and S.-C. Su, "A dynamic VPN architecture for private cloud computing," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput.*, Dec. 2011, pp. 409–414.

[34] L. Malina, G. Srivastava, P. Dzurenda, J. Hajny, and R. Fujdiak, "A secure publish/subscribe protocol for Internet of Things," in *Proc. 14th Int. Conf. Availability, Rel. Secur.*, Aug. 2019, pp. 1–10.

[35] K. An, S. Pradhan, F. Caglar, and A. Gokhale, "A publish/subscribe middleware for dependable and real-time resource monitoring in the cloud," in *Proc. Workshop Secure Dependable Middleware Cloud Monitor. Manage. (SDMCMM)*, 2012, pp. 1–6.

[36] 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," *Ann. Telecommun.*, vol. 72, nos. 5–6, pp. 297–305, 2017.

[37] 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 Commun. Mobile Comput.*, vol. 2021, pp. 1–11, Jun. 2021.

[38] V. Chvatal, "A greedy heuristic for the set-covering problem," *Math. Oper. Res.*, vol. 4, no. 3, pp. 233–235, Aug. 1979.

[39] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, Jul. 1972.

[40] A. Krause and D. Golovin, "Submodular function maximization," *Tractability, Pract. Approaches Hard Problems*, vol. 3, no. 19, pp. 71–104, 2012.

[41] G. L. Nemhauser and L. A. Wolsey, "Best algorithms for approximating the maximum of a submodular set function," *Math. Oper. Res.*, vol. 3, no. 3, pp. 177–188, 1978.

[42] H. Xu *et al.*, "Joint route selection and update scheduling for low-latency update in SDNs," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 3073–3087, Oct. 2017.

[43] R. Gandhi *et al.*, "Duet: Cloud scale load balancing with hardware and software," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 27–38, 2015.

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

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

[46] M.-W. Park and Y.-D. Kim, "A systematic procedure for setting parameters in simulated annealing algorithms," *Comput. Oper. Res.*, vol. 25, pp. 207–217, Mar. 1998.

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

[48] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas. Conf. (IMC)*, 2009, pp. 202–208.

[49] GoDaddy. *GoDaddy: An American Publicly Traded Internet Domain Registrar and Web Hosting Company*. Accessed: Jul. 20, 2021. [Online]. Available: https://www.godaddy.com/

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

[51] Apache kafka. *Apache Kafka: An Open-Source Distributed Event Streaming Platform*. Accessed: Jul. 20, 2021. [Online]. Available: https://kafka.apache.org/

[52] Apache Pulsar. *Apache Pulsar: Cloud-Native, Distributed Messaging and Streaming*. Accessed: Jul. 20, 2021. [Online]. Available: https://pulsar.apache.org/

[53] M. K. Sasubilli and R. Venkateswarlu, "Cloud computing security challenges, threats and vulnerabilities," in *Proc. 6th Int. Conf. Inventive Comput. Technol. (ICICT)*, Jan. 2021, pp. 476–480.

[54] A. M. Caulfield *et al.*, "A cloud-scale acceleration architecture," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–13.

[55] B. Jana, M. Chakraborty, and T. Mandal, "A task scheduling technique based on particle swarm optimization algorithm in cloud environment," in *Soft Computing: Theories and Applications* (Advances in Intelligent Systems and Computing). Berlin, Germany: Springer, 2019, pp. 525–536.

[56] R. Khorsand and M. Ramezanpour, "An energy-efficient task-scheduling algorithm based on a multi-criteria decision-making method in cloud computing," *Int. J. Commun. Syst.*, vol. 33, no. 9, p. e4379, Jun. 2020.

[57] F. Abazari, M. Analoui, H. Takabi, and S. Fu, "MOWS: Multi-objective workflow scheduling in cloud computing based on heuristic algorithm," *Simul. Model. Pract. Theory*, vol. 93, pp. 119–132, May 2019.

[58] R. Gouareb, V. Friderikos, and A.-H. Aghvami, "Virtual network functions routing and placement for edge cloud latency minimization," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2346–2357, Oct. 2018.

[59] H. Riasudheen, K. Selvamani, S. Mukherjee, and I. R. Divyasree, "An efficient energy-aware routing scheme for cloud-assisted MANETs in 5G," *Ad Hoc Netw.*, vol. 97, Feb. 2020, Art. no. 102021.

[60] R. C. Poonia and L. Raja, "On-demand routing protocols for vehicular cloud computing," in *Research Anthology on Architectures, Frameworks, and Integration Strategies for Distributed and Cloud Computing*. IGI Global, 2021, pp. 96–122.

[61] J. Kumar, A. K. Singh, and R. Buyya, "Self directed learning based workload forecasting model for cloud resource management," *Inf. Sci.*, vol. 543, pp. 345–366, Jan. 2021.

[62] S. Ramamoorthy, G. Ravikumar, B. S. Balaji, S. Balakrishnan, and K. Venkatachalam, "MCAMO: Multi constraint aware multi-objective resource scheduling optimization technique for cloud infrastructure services," *J. Ambient Intell. Humanized Comput.*, vol. 12, no. 6, pp. 5909–5916, Jun. 2022.

[63] J. N. Witanto, H. Lim, and M. Atiquzzaman, "Adaptive selection of dynamic VM consolidation algorithm using neural network for cloud resource management," *Future Gener. Comput. Syst.*, vol. 87, pp. 35–42, Oct. 2018.

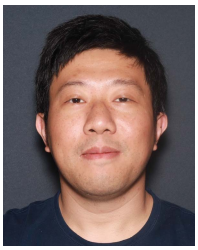
[64] D. T. Vojnak, B. S. Đorđević, V. V. Timčenko, and S. M. Štrbac, "Performance comparison of the type-2 hypervisor VirtualBox and VMWare workstation," in *Proc. 27th Telecommun. Forum (TELFOR)*. IEEE, 2019, pp. 1–4.
[65] P. Barham *et al.*, "Xen and the art of virtualization," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Dec. 2003.
[66] M. R. Shahriar, S. M. N. A. Sunny, X. Liu, M. C. Leu, L. Hu, and N.-T. Nguyen, "MTComm based virtualization and integration of physical machine operations with digital-twins in cyber-physical manufacturing cloud," in *Proc. 5th IEEE Int. Conf. Cyber Secur. Cloud Comput. (CSCloud), 4th IEEE Int. Conf. Edge Comput. Scalable Cloud (EdgeCom)*, Jun. 2018, pp. 46–51.
[67] Y. Zhang *et al.*, "KylinX: A dynamic library operating system for simplified and efficient cloud virtualization," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2018, pp. 173–186.
[68] H.-Y. Tsai, M. Siebenhaar, A. Miede, Y. Huang, and R. Steinmetz, "Threat as a service?: Virtualization's impact on cloud security," *IT Prof.*, vol. 14, no. 1, pp. 32–37, Jan./Feb. 2011.
[69] S. S. Manikandasaran, K. Balaji, and S. Raja, "Infrastructure virtualization security architecture specification for private cloud," *Int. J. Comput. Sci. Eng.*, vol. 6, no. 2, pp. 10–14, 2018.
[70] W. Pan, Y. Zhang, M. Yu, and J. Jing, "Improving virtualization security by splitting hypervisor into smaller components," in *Proc. Annu. Conf. Data Appl. Secur. Privacy*. Berlin, Germany: Springer, 2012, pp. 298–313.
[71] R. Kumar, K. Jain, H. Maharwal, N. Jain, and A. Dadhich, "Apache cloudstack: Open source infrastructure as a service cloud computing platform," *Proc. Int. J. Advancement Eng. Technol., Manage. Appl. Sci.*, vol. 111, p. 116, Jul. 2014.
[72] D. Nurmi *et al.*, "The eucalyptus open-source cloud-computing system," in *Proc. 9th IEEE/ACM Int. Symp. Cluster Comput. Grid*, 2009, pp. 124–131.
[73] D. Milojičić, I. M. Llorente, and R. S. Montero, "OpenNebula: A cloud management tool," *IEEE Internet Comput.*, vol. 15, no. 2, pp. 11–14, Mar. 2011.
[74] A. Gheith *et al.*, "IBM bluemix mobile cloud services," *IBM J. Res. Develop.*, vol. 60, nos. 2–3, pp. 1–7, 2016.
[75] B. Satzger, W. Hummer, C. Inzinger, P. Leitner, and S. Dustdar, "Winds of change: From vendor lock-in to the meta cloud," *IEEE Internet Comput.*, vol. 17, no. 1, pp. 69–73, Jan. 2013.
[76] N.-L. Tran, S. Skhiri, and E. Zim, "EQS: An elastic and scalable message queue for the cloud," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, Nov. 2011, pp. 391–398.
[77] Y. Kim and J. Park, "Hybrid decentralized PBFT blockchain framework for OpenStack message queue," *Hum.-Centric Comput. Inf. Sci.*, vol. 10, no. 1, pp. 1–12, Dec. 2020.
[78] T. Li, K. Keahey, K. Wang, D. Zhao, and I. Raicu, "A dynamically scalable cloud data infrastructure for sensor networks," in *Proc. 6th Workshop Sci. Cloud Comput.*, Jun. 2015, pp. 25–28.

Gongming Zhao (Member, IEEE) received the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2020. He is currently an Associate Professor with the University of Science and Technology of China. His current research interests include software-defined networks and cloud computing.

Luyao Luo (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree in computer science with the University of Science and Technology of China. His main research interests are software-defined networks and cloud computing.

Hongli Xu (Member, IEEE) received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China (USTC), China, in 2002 and 2007, respectively. He is currently a Professor with the School of Computer Science and Technology, USTC. He has published more than 100 articles in famous journals and conferences, including IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the International Conference on Computer Communications (INFOCOM), and the International Conference on Network Protocols (ICNP). He has held more than 30 patents. His research interests include software-defined networks, edge computing, and the Internet of Things. He was awarded the Outstanding Youth Science Foundation of NSFC in 2018. He has won the best paper award or the best paper candidate in several famous conferences.

Chun-Jen Chung received the M.S. degree in computer science from New York University and the Ph.D. degree in computer science from Arizona State University. He is a Senior Architect with the Cloud Laboratory, Futurewei Technologies Inc. His research interests include cyber security, cloud security, cloud networking, software-defined networking, cloud computing, and trust computing.



Liguang Xie (Senior Member, IEEE) received the Ph.D. degree in computer engineering from Virginia Tech, Blacksburg, VA, USA, in 2013. He is a Sr. Principal Architect and the Sr. Director of Engineering with Futurewei's Seattle Research Center, where he leads the Cloud Networking Research and Development Team and oversees cloud networking open-source, research, and engineering efforts. Prior to Futurewei, he was a Senior Software Engineering Lead with Microsoft Azure Networking and an Adjunct Assistant Professor with the Bradley Department of Electrical and Computer Engineering, Virginia Tech. His research interests are cloud networking, software-defined networking, distributed systems, edge computing, and cloud computing.