Dynamic Scaling of Virtualized, Distributed Service Chains: A Case Study of IMS

Jingpu Duan, Chuan Wu, Franck Le, Alex X. Liu, Yanghua Peng

***Abstract*—The emerging paradigm of network function virtualization advocates deploying virtualized network functions (VNFs) on standard virtualization platforms for significant cost reduction and management flexibility. There have been system designs for managing dynamic deployment and scaling of VNF service chains within one cloud datacenter. Many real-world network services involve geo-distributed service chains, with prominent examples of mobile core networks and IP Multimedia Subsystems (IMSs)). Virtualizing these service chains requires efficient coordination of dynamic VNF deployment across geo-distributed datacenters, calling for a new management system. This paper designs a dynamic scaling system for geo-distributed VNF service chains, using the case of an IMS. IMSs are widely used subsystems for delivering multimedia services among mobile users in a 3G/4G network, whose virtualization has been broadly advocated in the industry for reducing cost, improving network usage efficiency and enabling dynamic network topology reconfiguration for performance optimization. Our scaling system design caters to key control-plane and data-plane service chains in an IMS, combining proactive and reactive approaches for timely, cost-effective scaling of the service chains. The design principles are applicable to scaling of other systems with multiple related service chains. We evaluate our system using real-world experiments on both an emulation platform and a geo-distributed public cloud.**

***Index Terms*—Software defined networking, network function virtualization, IP multimedia subsystem.**

I. Introduction

Traditional hardware-based network functions are notoriously hard and costly to deploy and scale. The recent paradigm of network function virtualization (NFV) advocates deploying software network functions in virtualized environments (e.g., VMs) on off-the-shelf servers, to significantly simplify deployment and scaling at much lowered costs [1].

Despite the advantages, many problems remain when introducing NFV to the provisioning of practical network services. One problem is to design efficient VNF software, such that software VNFs can achieve packet processing speeds close to hardware middleboxes. Another is to design an efficient management system, which deploys and scales VNF service chains – an ordered collection of VNFs that altogether compose a network service, according to the traffic demand. There have been efforts targeting architectural improvement of VNF software [2]. A number of management systems have also been proposed [3], [4], which operate VNF service chains deployed in a single server cluster or datacenter. These management systems are adequate for service chains such as "firewall→ intrusion detection system (IDS)", which are typically used to provide access service to a client-server Web system, deployed in the on-premise cluster/datacenter of the service provider.

There are many other service chains which render a geo-distributed nature, e.g., the service chains in IP Multimedia Subsystems (IMS) [5] and mobile core networks [6] (examples presented in Fig. 1). In these systems, the network functions are desirably deployed close to geo-dispersed users, and putting the service chain in a single datacenter would be unfavourable as compared to distributing its VNFs across several datacenters. The existing management systems cannot be directly applied to handle such geo-distributed service chains [7], due to the escalated challenges on efficient interconnection of VNFs over the WAN, dynamic decision making on how VNF instances are deployed in different datacenters, and optimally dispatching individual flows through the deployed instances.

This paper presents *ScalIMS*, a management system that enables dynamic deployment and scaling of VNF service chains across multiple datacenters, using representative control-plane and data-plane service chains of the IMS system [5]. *ScalIMS* is designed to provide good performance (minimal VNF instances deployment and guaranteed end-to-end flow delays), using both runtime statistics of VNFs and global traffic information. IMS is chosen as the target platform because of its important role in the telecom core networks as well as the accessibility of open-source software implementation of IMS [8]. *ScalIMS* has two important characteristics that distinguish itself from existing management systems:

▹ **Dynamic Scaling over Multiple Datacenters:** *ScalIMS* dynamically deploys multiple instances of the same network function onto different datacenters according to real-time traffic demand and user distribution. The network paths that a service chain traverses are optimized to provide QoS guarantee of user traffic (i.e., bounded end-to-end delays). This feature distinguishes *ScalIMS* from systems that can only scale service chains within a single datacenter [3], [4].

▹ **A Hybrid Scaling Strategy:** Most existing VNF management systems [9] [4] scale service chains using reactive approaches, adding/removing VNF instances by responding to changes of runtime status of existing VNFs. Novelly, *ScalIMS* combines reactive scaling with proactive scaling, using predicted traffic volumes based on the history. This hybrid strategy exploits all opportunities for timely scaling of VNFs and significantly improves system performance.

Manuscript received April 10, 2017; accepted August 15, 2017.

J. Duan, C. Wu and Y. Peng are with the Department of Computer Science, The University of Hong Kong, email: (jpduan, cwu, yhpeng@cs.hku.hk).

F. Le is with the IBM T. J. Watson Research Center, email: (fle@us.ibm.com).

Alex X. Liu is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824 USA, and the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China (Email: alexliu@cse.msu.edu).

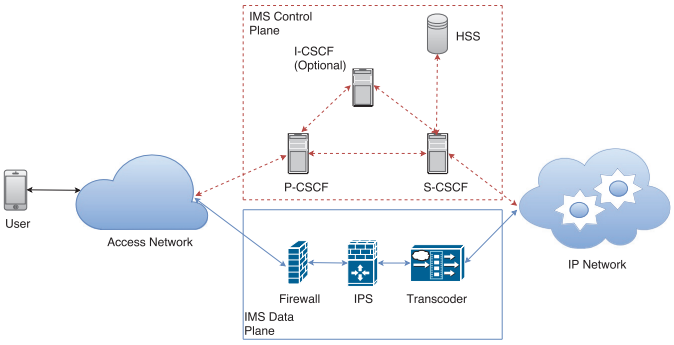

Fig. 1: IMS: an architectural overview

We evaluate *ScalIMS* on IBM SoftLayer cloud. Experiment results show that *ScalIMS* significantly improves QoS of user traffic compared with scaling systems that use only reactive or proactive scaling approaches. Meanwhile, *ScalIMS* achieves this improvement using almost 50% less VNF instances. Even though *ScalIMS* is designed for IMS systems, similar design principles can be easily applied to other NFV systems, which benefit from service chain deployment across multiple datacenters.

II. Background

A. IMS Overview

An IP Multimedia Subsystem (IMS) [5] is a core part in 3G/4G telecom networks (e.g., 3GPP, 3GPP2) [10] [11], responsible for delivering multimedia services (e.g., voice, video, messaging) over IP networks. It is a complex system consisting of multiple service chains. We investigate two most important service chains as follows. An illustration is given in Fig. 1.

▹ **Control Plane (CP) Service Chain** includes three main network functions, Proxy-CSCF (P-CSCF), Interrogating-CSCF (I-CSCF), and Serving-CSCF (S-CSCF), which collectively handle user registration, user authentication and call setup. These network functions rely on the Session Initiation Protocol (SIP) [12] to interoperate with users of the IMS system. Users can only contact with P-CSCF, which acts as a relay point between users and S-CSCF. Since I-CSCF acts as a middleman that forwards SIP messages between P-CSCF and S-CSCF, real-world implementation sometimes merges I-CSCF into S-CSCF as in [8] to simplify the structure of the IMS control plane service chain, making I-CSCF optional. S-CSCF dispatches SIP messages to their final destinations and constantly queries an external storage server called Home Subscriber Server (HSS), which is a database that contains identities of the users. We consider the control plane service chain P-CSCF → S-CSCF → P-CSCF in *ScalIMS*.

▹ **Data Plane (DP) Service Chain** contains a sequence of network functions that the actual multimedia traffic between users traverses, for security (e.g., firewall, deep packet inspection, intrusion detection), connectivity (e.g., NAT, IPv4-to-IPv6 conversion), quality of service (e.g., traffic shaping, rate limiting, ToS/DSCP bit setting), and media processing (e.g., transcoding). While 3GPP has standardized the IMS control plane for interoperability reasons, the exact set of deployed network functions for the data plane varies per operator.

The two service chains collectively handle two important procedures of the IMS system, which are user registration and call setup. To make a call, a user first registers his IP address to the IMS by initiating a SIP REGISTRATION transaction over the CP. When the registration is done, S-CSCF temporarily stores the binding between the identity of the user and the P-CSCF instance connected to the user for future calls. To setup a call between a caller and a callee, the caller initiates a SIP INVITE transaction to the IMS, specifying the identify of the callee. S-CSCF uses the binding saved during user registration to retrieve the P-CSCF instance that the callee connects to and sends the message to the P-CSCF instance, which forwards to the callee. After the callee responds, a call is successfully set up. Subsequent media flows between the caller and the callee are routed through DP service chain. When the call is finished, a SIP BYE transaction between the caller and the callee is carried out to close the call over the CP.

B. Related Work

Running VNF software (e.g., DP packet processing software) on VMs incurs significant context switching cost [13], limiting the maximum throughput of a VNF. To solve this problem, ClickOS [2] maps packets directly from NIC receive queues to a shared memory region, and fetch packets directly from that shared memory region [14], which greatly improve packet processing throughput. However, this approach completely by-passes the existing kernel networking stack, unable to support VNFs (e.g., S-CSCF and P-CSCF) that use the traditional TCP/IP stack.

Scaling of service chains has been investigated in a single server, a computing cluster or a datacenter. CoMB [15] focus on scaling VNFs in a single server, by designing customized architecture to unify VNFs inside a single server. E2 [3] scales VNF service chains in a single datacenter, exploiting high-performance inter-VNF data paths through SDN-enabled switches. Stratos [4] jointly consider VNF placement and flow distribution within a datacenter, using on-demand VNF provisioning and VM migration to mitigate hotspots.

The management systems mentioned above cannot be directly extended to the multi-datacenter setting. One primary reason is that SDN controllers [16] are extensively used in these systems to facilitate routing, scaling and load-balancing within a datacenter. However, SDN controllers are rarely available in the WAN, except for among datacenters of a few large providers such as Google [17] and Microsoft [18]. *ScalIMS* is a NFV management system that efficiently coordinates service chain deployment and scaling, as well as flow routing, across multiple data centers. *ScalIMS* uses similar methodologies as in [3] and [4] when scaling NFV service chains within a datacenter, but adopts a novel distributed flow routing approach and a proactive scaling strategy to scale NFV service chains across multiple datacenters.

Similar with *ScalIMS*, Klein [7] also scales NFV service chains across multiple datacenters. However, it focuses on scaling EPC system [6] and does not use a hybrid scaling strategy as *ScalIMS* does. Ren et al. [19] propose a VNF dynamic auto scaling algorithm for 5G networks, but it lacks a real-world implementation when compared to *ScalIMS*.

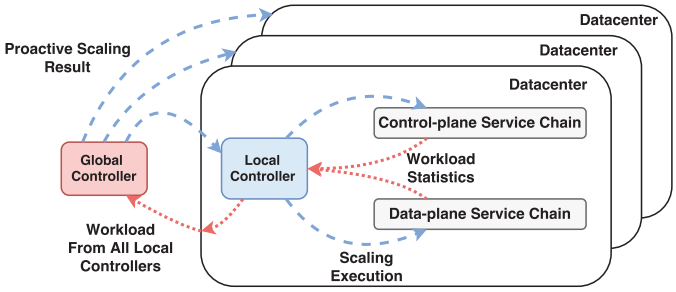

Fig. 2: Functional overview of *ScalIMS*

III. Challenges and Design Highlights

ScalIMS aims to address the following challenges, that arise when scaling service chains over multiple data centers.

First, deciding service chain paths, as determined by the datacenters where instances of VNFs in a service chain should be deployed. The service chain path critically decides service quality of user traffic along the chain. For instance, a traffic flow sent by a user of the IMS system to another user may have two optional paths. The first path traverses a sequence of datacenters (a, c) while the second path traverses datacenters (a, b, c). The end-to-end delays on the two paths may vary over time. A multi-datacenter NFV scaling system should constantly update the service chain paths, so that a good service quality can be guaranteed for user traffic at all time.

Second, deciding scaling in/out of network functions, i.e., adding/removing instances of each VNF upon traffic rise/drop. This decision is coupled with service chain path selection in the multi-datacenter setting. If a service chain path is overloaded, instead of launching new VNF instances on the same datacenters, the system may search for available VNF instances on other datacenters, and set up new service chain paths using those instances.

Third, distributed flow routing. When a service chain path traverses multiple datacenters, it is difficult for a single controller to control the end-to-end route. When multiple SDN controllers are employed in different datacenters, they should work in coordination on constant updates of service chain paths, and correctly route user traffic towards destinations.

We make the following design decisions in *ScalIMS*. A functional overview of *ScalIMS* is given in Fig. 2.

▹ We adopt a hybrid scaling strategy, that combines proactive scaling and reactive scaling for both CP and DP service chains. We divide the system time into *scaling intervals*. At the end of each scaling interval, proactive scaling is invoked, which takes as input the predicted workload along each service chain, inter-datacenter latencies and the current VNF deployment (the numbers of instances of each VNF on each datacenter), and generates decisions on VNF scaling and service chain path deployment with bounded end-to-end delay simultaneously for the next scaling interval. Reactive scaling produces scaling decisions of each VNF based on runtime statistics of each instance within each data center. It compensates for the inaccuracy of workload prediction with proactive scaling, improving system performance under unpredicted traffic rate changes.

▹ *ScalIMS* enables a synergy of global and local controllers, to best execute the hybrid scaling strategy. The global controller runs on a standalone server. The local controllers are SDN controllers in each data center. For proactive scaling, the global controller coordinates with all local controllers: it collects statistics from each local controller, including CPU/memory usage and network traffic volume, runs the proactive scaling algorithm, generates scaling/deployment decisions, and broadcasts the decisions to local controllers. Each local controller executes the received decisions by launching new VNF instances and adjusting service chain paths. For reactive scaling, a local controller collects runtime statistics from each VNF instance running in its datacenter, and produces local, reactive scaling decision. For flow routing, a local controller uses flow tags and service chain paths received from the global controller to determine the VNF instances that a flow should traverse within its datacenter and be dispatched to in other datacenters.

▹ The architecture of *ScalIMS* follows ETSI NFV MANO framework [20], where the global controller closely resembles the NFV orchestrator and the local controller works as both VNF manager and virtual infrastructure manager. Though *ScalIMS* is designed for IMS systems, it can be easily adapted to handle other service chain systems, which provide user inter-connection services with users distributed over a large geographical span. For instance, *ScalIMS* can be adapted to manage the virtualized service chains in Evolved Packet Core (EPC) in 4G LTE network [6], by augmenting the CP service chain of EPC with an edge proxy that simulates the functionality of P-CSCF of IMS.

IV. Scaling of Control Plane Service Chain

A. Deployment and Entry Datacenter Binding

The CP service chain consists of P-CSCF and S-CSCF. We use a fixed placement strategy by deploying P-CSCF instances on every datacenter and S-CSCF instances on a fixed selected datacenter, such that the delay between the datacenter where we place S-CSCF instances and other datacenters falls within an acceptable range (the acceptable SIP transaction completion time is typically 250ms).

The rationale behind such a fixed placement strategy is the following. Even if we spread S-CSCF instances over different datacenters, each S-CSCF instance still needs to access a central HSS server and a memcached cluster to process most of the SIP transactions. This fact urges us to place S-CSCF instances together with the HSS server and memcached cluster in the same selected datacenter. Since P-CSCF instances act as relay points for user flows to access S-CSCF instances, it is desirable to place P-CSCF instances on every datacenter, to facilitate users' access to a P-CSCF instance on the closest datacenter. This fixed placement strategy also simplifies the routing of SIP messages along the CP service chain.

ScalIMS binds each user to the nearest datacenter according to his current geographical location, which is referred to as the user's *entry datacenter*. A user's CP and DP traffic can only enter and depart from the respective service chains from his entry datacenter. Such a datacenter binding is a natural design

choice as each user should be pinned to a unique P-CSCF instance on a specific datacenter when he uses the IMS [5].

To implement entry datacenter binding, a DNS server is maintained in *ScalIMS*. When a user queries the IP address of an available P-CSCF instance by sending out a DNS request, the DNS server maps the user's IP address contained in the DNS request to a geographical location by querying an IP-location database (*e.g.*, IP location finder [21]), referred to as the *location service*, and then assigns a datacenter that is closest to the user's current location as his entry datacenter.

When a call is initiated between user a and user b, user b's entry datacenter is also referred to as user a's *exit datacenter*. In *ScalIMS*, each pair of datacenters may form an entry-exit datacenter pair. CP workload and DP workload between each entry-exit datacenter pair are maintained in *ScalIMS*, which are the aggregate rates of traffic that callers associated with an entry datacenter send to callees bound to the exit datacenter, along the CP service chain and the DP service chain, respectively. The traffic rates among entry-exit datacenter pairs are used for traffic prediction and VNF instance provision.

B. Proactive Scaling

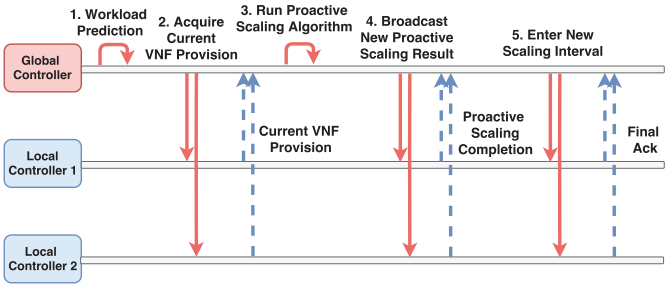

Fig. 3: Proactive scaling protocol.

Proactive scaling is executed in each scaling interval according to the protocol illustrated in Fig. 3.

1. Workload Prediction. Workload along a CP service chain is described by the number of SIP transactions carried out between each entry-exit datacenter pair every second. When a SIP transaction finishes, the S-CSCF instance involved uses the location service to determine which entry-exit pair this transaction belongs to (according to the IP addresses of the caller and the callee). Each S-CSCF instance keeps a record of the number of SIP transactions on each entry-exit datacenter pair and reports this number to the local controller in its datacenter every second. The local controller accumulates CP workload for 5 seconds before relaying it to global controller.

At the end of a scaling interval t, the global controller predicts the workload $\hat{u}_{t+1}$ in the next scaling interval using historic data in past several intervals (10 as in our experiments), using auto regression [9]: $\hat{u}_{t+1} = \mu + \phi(u_t - \mu)$. Here μ is the mean of the historic workload values in the past several scaling intervals, u_t is the average CP workload collected in the current interval, and ϕ can be decided using the covariance of the historical workload divided by the variance of the historical workload. The workload between each entry-exit datacenter pair is predicted this way.

2. Acquire Current VNF Provision. The global controller then broadcasts a message to local controllers, asking them to send the numbers of instances of each VNF provisioned in the respective datacenter. Upon receiving this request, a local controller knows that proactive scaling computation is on going, stops its reactive scaling process (Sec. IV-C) so that it does not interfere with proactive scaling, and then sends its current VNF provision information to the global controller.

3. Run Proactive Scaling Alrogihm. After receiving current VNF deployment from all local controllers, the global controller computes the numbers of P-CSCF and S-CSCF instances to be deployed in each datacenter in the next scaling interval. Since all S-CSCF instances are placed in the same datacenter, the number is decided by dividing the total predicted workload between all pairs of entry-exit datacenters by the processing capacity of S-CSCF. The number of P-CSCF instances to be deployed in a datacenter is computed by dividing the overall predicted workload for the entry-exit datacenter pairs, which use this datacenter as either entry or exit datacenter, by the processing capacity of P-CSCF.

4. Broadcast Proactive Scaling Result. The global controller then broadcasts the computed numbers to local controllers. A local controller sends a completion message to the global controller, after creating new VNF instances (scale-out) or enqueueing unused VNF instances to the respective buffer queues (scale-in), according to the received numbers.

5. Enter New Scaling Interval. After the global controller receives completion messages from all local controllers, it broadcasts an "enter new scaling interval" message to all local controllers. After receiving this message, a local controller increments its scaling interval index by 1, and shuts down some VNF instances from the head of the buffer queues. Then the local controller sends a final acknowledgement to the global controller. Upon receiving all final acknowledgements, the global controller increases its scaling interval index by 1.

Buffer Queue. In each datacenter, a double-ended buffer queue is maintained to temporarily hold unused VNF instances, with one queue for one type of VNF. When a VNF instance is to be removed, instead of directly shutting it down, the local controller tags it with the index of the current scaling interval, and enqueues it to the tail of the respective buffer queue. Once a VNF instance is enqueued, no more flows will be routed to it. Whenever more instances of a VNF are to be established, if there are available instances in the buffer queue of this VNF, buffered instances will be popped out from the tail of the queue, and transformed back to working instances, to fulfil the demand as much as possible. The purpose is to avoid creating new VNF instances frequently and improve flow loss rate, as the typical VNF creation time can last a few seconds and has a bad influence on flow loss rate. Unused buffered VNF instances are destroyed after τ scaling intervals (τ is set to 10 in our implementation), in Step 5 above.

C. Reactive Scaling

An agent running on each VNF instance reports to the local controller runtime statistics of the instance, e.g., CPU usage, memory usage and the number of input packets, in each second. The local controller maintains time series of these statistics for each VNF instance. It decides whether

CPU, memory, or network is overloaded during the past several seconds by comparing the respective statistics with a threshold (Sec. VI). If overload is persistently identified for at least two types of statistics (*e.g.*, CPU and network usage) for some consecutive time (5 seconds as in our experiments), then that VNF instance is reported as overloaded. We make the decision using two types of statistics in order to eliminate false alarms brought by examining a single statistics. The local controller then avoids routing new traffic flows (*i.e.*, new calls) to overloaded instances if there are other available instances. In a datacenter, if the states of a majority of instances of a VNF are "overloaded", scale-out is triggered by adding one new instance of that VNF. Note that no scale-in decisions (i.e., removing instances) are made reactively. They are solely handled by the proactive scaling protocol, as it improves system stability during workload fluctuation.

D. Flow Routing on Control Plane

When a user connects to the IMS system, he first issues a query to a DNS server, which determines the entry datacenter of this user and obtains the IP address of an available P-CSCF instance (non-overloaded) by querying the local controller of the entry datacenter. A P-CSCF instance learns the IP addresses of several available S-CSCF instances (non overloaded) by querying the local controller of the datacenter hosting S-CSCF instances and distributes its up-stream requests to these S-CSCF instances evenly. It regularly (every 30s in *ScalIMS*) updates the connections to up-stream S-CSCF instances by querying the local controller to obtain an updated view of S-CSCF instances.

A CP flow for establishment of a call (i.e., a SIP INVITE transaction) runs as follows. The caller sends out a SIP INVITE message, carrying caller's source IP address, receive port and send port, to the assigned P-CSCF instance. The P-CSCF instance forwards the message to one connected S-CSCF instance. The S-CSCF instance queries the HSS database to obtain the P-CSCF instance assigned to the callee that is saved when callee registers himself, and forwards the SIP INVITE message to that P-CSCF instance. The P-CSCF instance assigned to the callee modifies the source IP field in the message to an IP address located on the callee's entry datacenter, so that the callee can learn an IP address located on callee's entry datacenter. The P-CSCF instance then sends the modified SIP INVITE message to the callee. The callee responds with a SIP OK message, going through the same service chain in the reversed direction. When the SIP OK message passes through caller's entry datacenter, the P-CSCF instance modifies source IP field in the message to an IP address located on caller's entry datacenter as well. When such a SIP INVITE transaction ends, the caller and the callee use the learned IP addresses as the destination IP addresses of the DP media flows and send their media flows to their entry datacenters, from where the media flows enter the DP service chain.

Besides SIP message modification, a P-CSCF instance sends two mappings (Table I) to its local controller after it has received the SIP OK message. The local controller saves these mappings for use when processing the DP flows (Sec. V).

TABLE I: Mappings saved on local controller

Controller on Caller Entry DC	1. (caller IP, caller send port)→(callee IP)
	2. (callee IP, callee send port)→(an IP on caller's entry datacenter, caller IP, caller receive port)
Controller on Callee Entry DC	3. (callee IP, callee send port)→(caller IP)
	4. (caller IP, caller send port)→(an IP on callee's entry datacenter, callee IP, callee receive port)

V. Scaling of Data Plane Service Chain

The DP service chain adopts the same reactive scaling mechanism as discussed in Sec. IV-C. For proactive scaling, the same steps as shown in Fig. 3 is followed, with the following differences.

First, the proactive scaling algorithm for DP service chain, running in Step 3 in Fig. 3, not only decides how VNF instances are provisioned in each datacenter, but also updates the service chain path (Sec. V-A) between each entry-exit datacenter pair. The new proactive scaling algorithm will be discussed in Sec. V-B.

Second, workload along a DP service chain is described as the number of packets transmitted over each entry-exit datacenter pair every second. The local controller acquires input DP workload using workload measuring OpenFlow rules installed on the SDN switch at each datacenter, and reports DP workload measurements to the global controller every second. Local controllers also constantly measure inter-datacenter delays through a ping test among each other, and report the ping delays to the global controller every second. In Step 1 of Fig. 3, not only workload but also delays between datacenters are predicted for the next interval, using the same approach as discussed in Sec. IV-B.

Next, each local controller also acts as the SDN controller to manage DP flow routing within the respective datacenter. When a local controller receives DP proactive scaling results in Step 4 of Fig. 3, it immediately adds/removes DP VNF instances accordingly, but saves the new DP service chain paths and uses them for routing only after receiving the "enter new scaling interval" message in Step 5 of Fig. 3 (Sec. V-D).

A. DP Service Chain Path

ScalIMS employs one DP service chain path for all the DP media flows sharing the same entry-exit datacenter pair. A service chain path is a sequence of datacenters $l[0], \ldots, l[m+1]$, where $l[0]$ and $l[m+1]$ are the indexes of the entry datacenter and exit datacenter, respectively, and $l[i], 1 \leq i \leq m$, is the index of the datacenter hosting the ith VNF in a m-stage service chain. For example, the DP service chain in our implementation of *ScalIMS* is "firewall (stage 1)→IDS (stage 2) → transcoder (stage 3)", and a service chain path is a sequence of 5 datacenters. The DP proactive scaling algorithm constantly adjusts the service chain path for each entry-exit datacenter pair, to efficiently utilize deployed VNF instances.

The definition of service chain path augments an actual service chain with a virtual entry stage 0 and a virtual exit stage $m+1$. These two stages are forced to be placed on the entry and exit datacenter respectively, so that entry and exit datacenters are guaranteed to be connected together. The two virtual stages have infinite capacities.

Each service chain path should satisfy two conditions. (i) Looplessness: if datacenter i hosts both stages x and y, $x < y$

Algorithm 1: DP Proactive Scaling Algorithm

Input: Predicted delay between each datacenter pair, predicted workload of each entry-exit datacenter pair, current VNF deployment, current service chain paths
Output: New VNF instance provisioning, new service chain paths

1 Compute total available processing capacity of instances of each VNF in each datacenter;
2 **foreach** *entry-exit datacenter pair* p, $p.entry \neq p.exit$ **do**
3 **if** *there is enough VNF capacity on* p*'s current service chain path and the end-to-end delay threshold is not violated along* p*'s current path* **then**
4 use p's current service chain path as new path and reduce available processing capacities of VNFs on p's new path by predicted workload;
5 **foreach** p, $p.entry \neq p.exit$ && *p's new service chain path has not been determined* **do**
6 compute a new path for p using Alg. 2;
7 **if** *there is not enough capacity on* p*'s new path* **then**
8 **foreach** *datacenter* d *on the new path* **do**
9 create $\lceil \max\{0, Q - Q'\}/C \rceil$ new instances of the VNF that datacenter d hosts for p, where Q is p's predicted workload, Q' is the total capacity of the VNF in d and C is per-instance processing capacity of that VNF;
10 reduce available processing capacities of instances of the VNF in d by predicted workload;
11 **foreach** p, $p.entry = p.exit$ **do**
12 use p's current service chain path as new path, and carry out same actions as in line 7-10 ;
13 scale in by enqueuing un-used VNF instances to the respective buffer queues;

Algorithm 2: Service Chain Path Computation

Input: Predicted inter-datacenter delays, an entry-exit datacenter pair $p = (entry, exit)$, p's predicted workload, p's current service chain path, current available processing capacities of VNF instances, the service chain of m stages, n datacenters
Output: p's new service chain path

1 minProvPath = p's current path;
2 **for** $v = 0, \ldots, exit - 1, exit + 1, \ldots, n - 1$ **do**
3 $record[0] = entry$, $record[1] = v$, $record[m + 1] = exit$;
4 **for** $x = 2, \ldots, m$ **do**
5 find out datacenter v_1 ($v_1 \neq exit$) that has the largest available capacity for stage-x VNF;
6 $record[x] = v_1$;
7 **if** *there is path loop on* $record$ **then**
8 eliminate loop by adjusting $record$;
9 $path[0, \ldots, x] = record[0, \ldots, x]$;
10 $path[x + 1, \ldots, m + 1] = exit$;
11 **if** $path$ *leads to a smaller number of new VNF instances to be created for hanlding predicted workload than minProvPath and satisfies end-to-end delay requirement* **then**
12 minProvPath = path
13 $path[0] = entry$, $path[1] = v$, $path[2, \ldots, m + 1] = exit$;
14 check whether $path$ should be assigned to minProvPath as in lines 11-12;
15 $path1[0] = entry$, $path1[1, \ldots, m + 1] = exit$;
16 $path2[0, \ldots, m] = entry$, $path2[m + 1] = exit$;
17 check whether $path1$ or $path2$ should be assigned to minProvPath as in lines 11-12;
18 **if** *minProvPath violates end-to-end delay requirement* **then**
19 find out the shortest-delay datacenter path between entry and exit;
20 run stage placement algorithm in Alg. 3 and assign the identified service chain path to minProvPath;
21 **return** *minProvPath*;

on the service chain, then datacenter i must host stage z, where $x < z < y$ too; otherwise, a routing loop is created on the inter-datacenter network, which increases end-to-end delay and wastes important inter-datacenetr network bandwidth. There is no need for *ScalIMS* to tackle routing loops inside a datacenter, as routing loops inside datacentres are not common and can be resolved by method described in [22]. (ii) Bounded end-to-end delay between the entry datacenter and the exit datacenter along the service chain path, by a pre-defined threshold.

B. DP Proactive Scaling Algorithm

The DP proactive scaling algorithm is given in Alg. 1, which computes a new service chain path for the next scaling interval, for each entry-exit datacenter pair. The algorithm first tries to reuse as many existing service chain paths as possible based on the current VNF provisioning, as long as the capacity is sufficient to handle predicted workload and the end-to-end delay threshold is still guaranteed (lines 2-4). In case that an existing service chain path can not be reused, a new service chain path is computed using Alg. 2 (lines 5-6), and scale-out is carried out if there is a shortage of VNF processing capacities (lines 7-10). For an entry-exit datacenter pair p where the entry and exit datacenters are the same, the entire service chain path of p is always deployed in this datacenter (lines 11-12). Finally, scale-in is carried out to enqueue un-used VNF instances into the buffer queue (line 13).

C. Service Chain Path Computation

Algorithm Overview. Alg. 2 presents the algorithm to compute a good service chain path between a given entry-exit datacenter pair, that aims to minimize the number of new VNF instances to be created while satisfing the end-to-end delay requirement. For a service chain of m VNFs (stages) and n datacenters, exhaustive search to identify such a service chain path incurs $O(m^n)$ running time. Instead, Alg. 2 seeks to optimistically find a good path in $O(mn)$ time.

In Alg. 2, the $record$ list retains the service chain path under investigation (line 3). The search starts by looping through all datacenters except the exit datacenter, to decide the one for hosting instances of stage-1 VNF (lines 2-3). For each subsequent VNF in the service chain, a datacenter (except the exit datacenter) with the largest available processing capacity of the respective VNF is chosen (lines 4-6). We might have created a loop in the service chain path. If so, the loop is eliminated (lines 7-8) using the method to be discussed next. Whenever the datacenter to host stage-x VNF is determined, a candidate path is produced by assuming all the rest VNF stages $(x+1, \ldots, m)$ will be hosted in the exit datacenter (lines 9-10). The candidate path is examined by calculating the number of new VNF instances to be created along this path and the end-to-end delay of this path. If the candidate path incurs addition of fewer new VNF instances than the current best candidate path while satisfying end-to-end delay requirement, we retain it in $minProvPath$ (lines 11-12). The algorithm also checks some naive candidate paths that are not generated by the search loop (lines 13-17). Finally, it is possible that all candidate paths found so far fail to satisfy the delay requirement. If so, we compute the shortest end-to-end delay path using a shortest path algorithm, and run the stage placement algorithm in Alg. 3 to produce the service chain path (lines 18-20).

Algorithm 3: Stage Placement Algorithm

Input: The shortest-delay datacenter path $d[0...k-1]$ as a list of distinct datacenter indices between entry-exit datacenter pair $(d[0], d[k-1])$; DP service chain with $m+2$ stages (augmented with virtual entry and exit stage discussed in Sec. V-A) and per-instance capacity $C_j, 0 \le j \le m+1$, of stage-j VNF; overall processing capacities $Q'_{i,j}$ of all stage-j VNF instances in datacenter $d[i]$; predicted workload Q for entry-exit datacenter pair $(d[0], d[k-1])$

Output: service chain path $l[0], \ldots, l[m+1]$ for entry-exit datacenter pair $(d[0], d[k-1])$.

1 Calculate $N(i,j), \forall i = 0, \ldots, k-1, j = 0, \ldots, m+1$, as follows:

$$num(i,j) = \begin{cases} 0, & \text{if } Q'_{i,j} \ge Q \text{ and } m-j+1 \ge k-i-1 \\ \lceil (Q - Q'_{i,j})/C_j \rceil, & \text{if } Q'_{i,j} < Q \text{ and } m-j+1 \ge k-i-1 \\ +\infty, & \text{if } m-j+1 < k-i-1 \end{cases} \quad (1)$$

$$N(0,0) = 0, N(i,0) = +\infty, i = 1, \ldots, k-1 \quad (2)$$

$$N(0,j) = N(0,j-1) + num(0,j), j = 1, \ldots, m+1 \quad (3)$$

$$N(i,j) = min\{N(i-1,j-1) + num(i,j), N(i,j-1) + num(i,j)\}, \quad i = 1, \ldots, k-1, j = 1, \ldots, m+1 \quad (4)$$

2 Backtrack from $N(k-1, m+1)$ to derive the service chain path $l[0], \ldots, l[m+1]$;

3 **return** $l[0], \ldots, l[m+1]$;

Loop Elimination. To eliminate a loop in the path introduced in lines 5-6 of Alg. 2, we adjust the *record* list following two ways: (i) place all VNF stages involved in the loop on the datacenter at the start of the loop. Suppose the path with loop is $(1, 2, 4, 2, 5)$. It is adjusted to $(1, 2, 2, 2, 5)$, i.e., the datacenter hosting stage 2 is changed from datacenter 4 to datacenter 2. (ii) Choose a new datacenter to replace the datacenter that leads to the loop, which has the second largest available capacity for hosting the respective VNF and will not create another loop. Suppose datacenter 3 in the above example has the second largest capacity of stage-3 VNF. Then the path is adjusted to $(1, 2, 4, 3, 5)$. The *record* list is adjusted in both ways and the two resulting lists are compared. The list requiring fewer new VNF instances is selected.

Stage Placement Algorithm. Alg. 3 gives the stage placement algorithm used in line 20 of Alg. 2, which calculates a service chain path, i.e., the sequence of datacenters to host VNF stages on the service chain, that minimizes the number of new VNF instances to be created on the path.

In Alg. 3, $num(i,j)$ is the number of new stage-j VNF instances to be created, if stage j is to be deployed on datacenter $d[i]$. $num(i,j)$ is computed in Eqn. 1 based on the following observation: if datacenter $d[i]$ is chosen to host stage j, then the remaining number of yet-to-be-placed stages, $(m+2)-(j+1)$, must be no smaller than the remaining number of datacenters which do not host any stage yet, $k-(i+1)$, as otherwise the resulting service chain is not able to traverse the shortest-delay datacenter path in sequence.

$N(i,j)$ is the minimum number of instances of stage 0 to stage j VNFs, if stage j is to be deployed on datacenter $d[i]$. The computation of $N(i,j)$ is a dynamic programming problem (Eqn. (2) to (4) in Alg. 3). Eqn. (2) to Eqn. (3) are base cases, initialized according to the fact that stage 0 is on the entry datacenter $d[0]$. Eqn. (4) is derived based on the following observation: if datacenter $d[i]$ is chosen to host stage j, then stage $j-1$ can only be hosted on either datacenter $d[i-1]$ (the previous datacenter on the datacenter path) or datacenter $d[i]$.

When calculating $N(i,j)$ in Eqn. (4), we can construct a link connecting $N(i,j)$ to either $N(i-1,j-1)$ or $N(i,j-1)$, depending on whether $N(i-1,j-1)+num(i,j)$ or $N(i,j-1)+num(i,j)$ is smaller. This indicates whether datacenter $d[i-1]$ or $d[i]$ should host stage $j-1$, if stage j is to be deployed on datacenter $d[i]$. After $N(k-1,m+1)$ is computed (recall stage $m+1$ must be placed on the exit datacenter $d[k-1]$), we can backtrack to obtain the best service chain path.

D. Flow Routing On Data Plane

Each DP media flow enters the system from the entry datacenter, is routed through the datacenters on its service chain path in a fully distributed fashion, and then departs from the exit datacenter. The DP media flow sent by the caller is used as an example to explain the distributed routing process in the following paragraphs.

Enter Service Chain Path. The caller learns an IP address located in his entry datacenter when the SIP OK message is received (Sec. IV-D). This IP address is used as the destination IP address to send the DP media flow to the entry datacenter.

Route through Service Chain Path. The local controller in a datacenter decides the service chain path used by a media flow, when the first packet of the flow arrives at the datacenter and triggers an OpenFlow Packet_IN message at the local controller. The local controller examines whether the packet header contains a special tag. In our implementation of *ScalIMS*, the tag is added to the destination port field by a special OpenFlow rule in the flow's entry datacenter. The tag contains the indices of the flow's entry and exit datacenters, and the current scaling interval number modulo 4.

If the special tag is not present, it indicates that this datacenter is the entry datacenter of the flow. Then the local controller uses the mapping 1 in Table I, saved during the SIP INVITE transaction, to get the callee IP address and obtains the index of the flow's exit datacenter using the location service (Sec. IV). The local controller then selects the service chain path corresponding to the flow's entry-exit datacenter pair, recorded for the current scaling interval. If there is one or multiple VNF stages hosted in this entry datacenter, the local controller selects a sequence of VNF instances for those VNF stages, according to a smallest workload-first principle for load balancing. Next, it installs several OpenFlow rules on the SDN switches in the datacenter, to route the flow along the selected sequence of VNF instances. The installed OpenFlow rule will also add the special tag to the header of all incoming packets of the flow. If the datacenter is not the exit datacenter, the flow is then routed to the next datacenter on the service chain path, through a VxLAN tunnel, that is set up between each pair of datacenters. Each VxLAN tunnel connecting a pair of datacenters is constructed over the inter-datacenter network connecting that pair of datacenters. Different service chain paths share the same VxLAN tunnel if they need to traverse the same pair of datacenters.

If the special tag is present, the datacenter hosts VNF stage(s) on the service chain path. The local controller selects

the service chain path indicated in the tag. Then it selects VNF instance(s) for the VNF stage(s), installs OpenFlow rules to route the flow through the instance(s), and routes flow out to the next datacenter (if it is not the exit datacenter), in the same way as discussed in the previous case. The use of the special tag ensures that each flow is routed through a consistent service chain path.

Exit from Service Chain Path. At the exit datacenter, the local controller uses mapping 4 in Table I to retrieve the IP address on callee's entry datacenter, callee IP and callee's receive port. If the exit datacenter also hosts VNF stage(s) in the service chain, the flow is routed through the VNF instance(s). Then the local controller uses an OpenFlow rule to perform an address translation, which replaces the flow's source IP address by the IP address on callee's entry datacenter, flow's destination IP address by the callee IP, and flow's destination port by the callee's receive port. The flow is then delivered to the callee over the Internet.

E. Handling Scaling Interval Inconsistency

In Step 5 of Fig. 3, the global controller sends an 'enter new scaling interval' message to all local controllers to advance the scaling interval index on each local controller by 1. Due to network delay, the local controllers may not receive the message at the same time, resulting in temporary inconsistency of scaling interval indices on different local controllers.

When a local controller is processing a flow, it may find that the encoded scaling interval in the header of the received flow packets does not match its current scaling interval index: one scaling interval ahead, or one scaling interval behind. In the first case, since the service chain paths for the next scaling interval are broadcast and saved at all local controllers before the 'enter new scaling interval' message can be sent and received by any local controller, the current local controller must have received the service chain paths for the next scaling interval, though not yet receiving the 'enter new scaling interval' message; it can use the service chain path for the next scaling interval to route the flow. In the second case, the local controller still uses the service chain path for the previous scaling interval to route the flow.

Since the difference between the scaling interval in the tag of received flow packets and the scaling interval on a local controller is among $-1, 0$, and 1, the scaling interval encoded in the tag is the actual value modulo 4, to reduce the number of bits required to only 2.

VI. Implementation and Evaluation

A. Implementation

We implement a prototype of *ScalIMS* in Java and deploy *ScalIMS* code on both local controller and global controller. The local controller is implemented as a module in the FloodLight SDN controller [23]. The global controller is implemented as a multi-threaded server program, which communicates with local controllers over regular sockets. We also implement a traffic generator based on PJSIP [24]. We deploy one traffic generator in each datacenter, producing user arrivals bound to the datacenter at a configurable rate. A global traffic generator coordinator receives notifications of generated users and pairs up users in a first-come-first-match manner. A call process is launched between every pair of paired users, which includes SIP transactions to establish a call, followed by a one-minute voice call at the bit rate of 80kbit/s, as well as necessary SIP transactions to shutdown the call.

TABLE II: VNF Capacity and Overload Threshold

VNF	Capacity	CPU threashold	Memory threshold	Input pkts/s threshold
P-CSCF	500 tran/s	70%	50%	1000 pkts/s
S-CSCF	200 tran/s	70%	50%	400 pkts/s
Firewall	35000 pkts/s	90%	50%	35000 pkts/s
IDS	20000 pkts/s	90%	50%	20000 pkts/s
Transcoder	15000 pkts/s	90%	50%	15000 pkts/s

We use P-CSCF, S-CSCF and HSS components from the Project Clearwater [8] as our CP VNFs. I-CSCF is omitted by *ScalIMS* as I-CSCF is optional and merged into S-CSCF by Project Clearwater. The DP service chain contains a firewall (implemented using user space Click [2]), an intrusion detector (Snort IDS [25]) and a transcoder (implemented using user space Click). Each network function runs on a QEMU/KVM VM. A VM to run a CP VNF is configured with 1 core and 2GB RAM. A VM to run a DP VNF is configured with 2 cores and 2GB RAM. The capacity and overload threshold (to decide scale-out) of each instance of each VNF are given in Table II, which are obtained by stress testing VNF instances to overloaded states.

B. Evaluation in IBM SoftLayer Cloud

We evaluate the performance of *ScalIMS* on IBM SoftLayer Cloud [26] by renting one bare-metal server in each of the 4 Softlayer datacenters, located in Tokyo, Hong Kong, London and Houston, respectively. Each server is equipped with two 6-core 2.4GHz Intel CPU, 64GB RAM, and 1TB SATA disk. In the SoftLayer cloud, servers in different datacenters are connected through a global private network [26], which provides a Gigabyte throughput. *ScalIMS* creates a VxLAN tunnel mesh in the private network to route DP traffic. CP VNFs are connected directly over the private network. The difference in the connection method is because CP VNFs are addressable at L3 layer (IP layer) whereas DP VNFs are only addressable at L2 layer (Ethernet layer).

We evaluate the performance of *ScalIMS* based on two groups of metrics. (1) The total number of new VNF instances created over time: the smaller the number is, the more cost/resource effective *ScalIMS* is; (2) QoS of user traffic including the SIP transaction completion time for CP flows, the RTT and loss rate for DP flows: a large number of QoS data is collected and their cumulative distribution function (CDF) curves are shown (as in Fig. 5, 6, 7), so that the higher the CDF curve is on a figure, the better the QoS is.

We compare the performance achieved by using proactive scaling only (the local controller does not react to overload of instances), reactive scaling only (the global controller initializes a good set of service chain paths, but no subsequent proactive scaling decisions are sent to local controllers), and both proactive and reactive scaling enabled (*i.e.*, the combined scaling strategy of *ScalIMS*). The reactive-scaling-only is the base-line case as the reactive scaling used by *ScalIMS* inside a single datacenter is similar to that of [4], [3]. Each scaling interval is set to be 50 seconds long, and each buffer

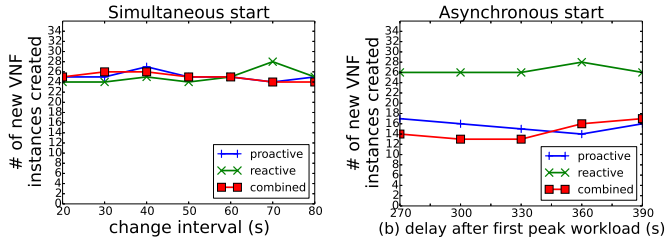

Fig. 4: Overall number of new VNF instances created: (a) simultaneous start; (b) asynchronous start.

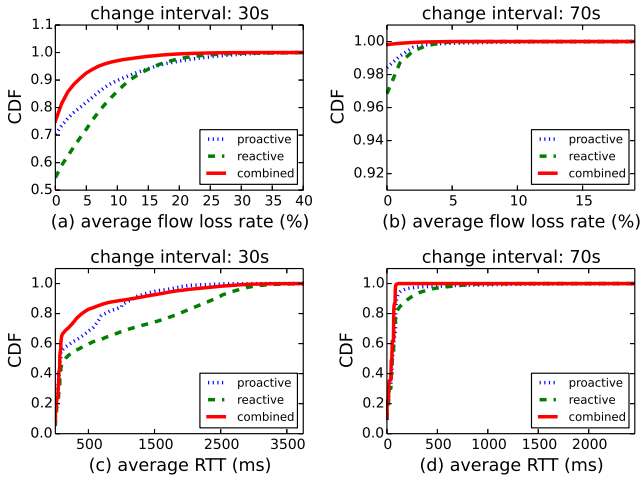

Fig. 5: QoS of DP flows: simultaneous start.

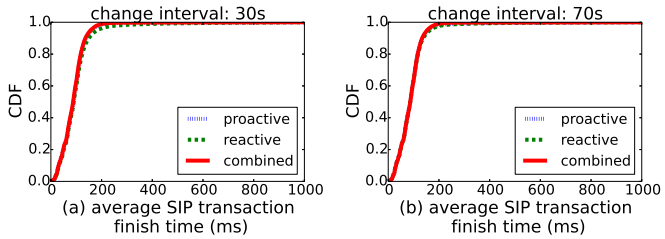

Fig. 6: QoS of CP flows: simultaneous start.

queue retains an unused VNF instance for at most 10 scaling intervals. The maximum allowed end-to-end flow delay is 250ms.

1) Simultaneous Start: In this set of experiments, each traffic generator starts generating users simultaneously at the rate of 1 $users/s$, which gradually increases to 15 $users/s$ and then decreases to 6 $users/s$. The rate change happens once every *change interval*. The duration of change intervals is set to different values in different experiments. In this way, the peak workload arrives at each datacenter almost concurrently.

Fig. 4(a) shows that the total number of VNF instances provisioned (for both CP and DP service chains) does not differ much under the three schemes. Since the maximum workload on each datacenter arrives at around the same time, the proactive scaling algorithm has no opportunity to decrease the number of provisioned VNF instances by adjusting the service chain path, and therefore the numbers are similar.

Fig. 5 plots CDFs of the number of DP flows at different average packet loss rates and average RTTs, and shows that the combined scaling strategy of *ScalIMS* out-performs the

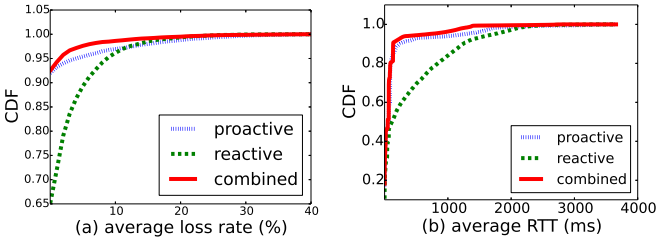

Fig. 7: QoS of DP flows: asynchronous start.

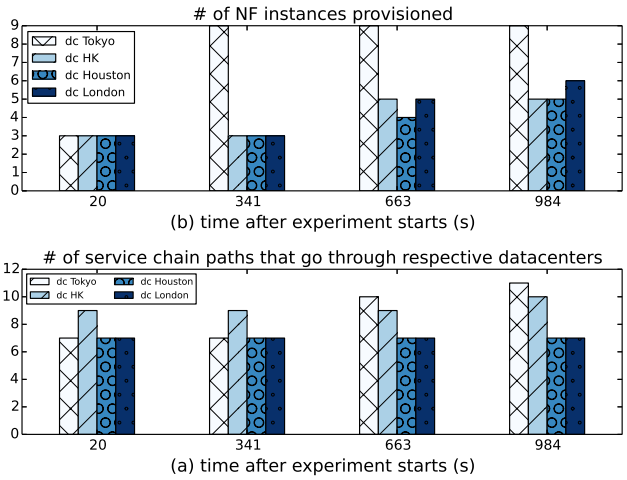

Fig. 8: Number of VNF instances in/service chain paths through each datacenter: asynchronous start.

other two strategies in terms of DP traffic quality. Pure reactive scaling adds new instances only when overload occurs. During the boot-up time of new instances, traffic continues to arrive at the overloaded VNF instances, resulting in a high packet loss rate and then high RTT. Pure proactive scaling adjusts VNF instances once every scaling interval. During each scaling interval, increased workload may have overloaded the system. The best performance is achieved combining both proactive and reactive scaling.

Fig. 6 shows that the average SIP transaction completion time is similar with the three schemes. This is because scaling of CP service chains is not triggered as often as DP service chains, since each instance of a CP VNF is able to handle a large number of SIP transactions.

2) Asynchronous Start: In this set of experiments, the traffic generator in the Tokyo datacenter is started first, followed by the traffic generators in Hong Kong, in Houston and then in London. In this way, the peak workload in different datacenters does not occur at the same time. Each traffic generator increases its user generation rate from 5 $users/s$ to 15 $users/s$ and then reduces it to 5 $users/s$, with a 30s change interval.

In Fig. 4(b), the start delay between traffic generators in Tokyo and in Hong Kong is set according to the values on the x axis, while start delays between traffic generators in Hong Kong and in Houston, and between traffic generators in Houston and in London, are set to 120s. We observe that the number of VNF instances created with the combined scaling approach is similar to proactive scaling approach, but always

smaller than reactive scaling approach. Fig. 7 shows that the traffic quality is the best with the combined approach as well.

Why can the combined scaling strategy perform well even when it creates a smaller number of VNF instances? The Tokyo datacenter sees the peak workload first, leading to the provisioning of many VNF instances in the datacenter, which later become redundant and will be buffered for 10 scaling intervals. When peak workload arrives at other datacenters, the global controller can re-use the buffered VNF instances by creating service chain paths traversing the Tokyo datacenter. These phenomena are exhibited in Fig. 8.

The performance of CP SIP transaction completion time under asynchronous start is very similar to the results presented in Fig. 6, and the figures are omitted due to space limit.

VII. Conclusion

This paper proposes *ScalIMS*, a NFV management system designed to deploy and scale service chains spanning geo-distributed datacenters, using the case of an IMS. *ScalIMS* features joint proactive and reactive scaling of DP and CP service chains, for timely and cost-effective provisioning of practical network services. Evaluation of our prototype implementation on IBM SoftLayer cloud shows that: (i) *ScalIMS* improves QoS of user traffic by a large margin when compared with pure reactive scaling; (ii) when peak workload arrives asynchronously over the geographic span, *ScalIMS* effectively reduces the total number of VNF instances provisioned while guaranteeing excellent QoS.

Acknowledgment

This work was supported in part by grants from Hong Kong RGC (17204715, 17225516, C7036-15G), Huawei HIRP (HO2016050002BE). This work was also supported by National Science Foundation under Grant Numbers CNS-1318563, CNS-1524698, and CNS-1421407, the National Natural Science Foundation of China under Grant Numbers 61472184 and 61321491, and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.

References

[1] "NFV," http://www.3gpp.org/DynaReport/23228.htm.
[2] M. Joao, A. Mohamed, and R. Costin, "Clickos and the art of network function virtualization," in *NSDI*, 2014.
[3] P. Shoumik, L. Chang, and H. Sangjin, "E2: a framework for nfv applications," in *SOSP*, 2015.
[4] G. Aaron, G. Robert, and A. Ashok, "Stratos: Virtual middleboxes as first-class entities," *UW-Madison TR1771*, 2012.
[5] "3GPP specification: 23.228," http://www.3gpp.org/ftp/Specs/html-info/23228.htm.
[6] "EPC," http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core.
[7] Q. Z. Ayyub, P. P. Krishna, and S. Vyas, "Klein: A minimally disruptive design for an elastic cellular core," in *SOSR*, 2016.
[8] "Project Clearwater," http://www.projectclearwater.org/.
[9] W. Timothy, S. Prashant, and V. Arun, "Black-box and gray-box strategies for virtual machine migration." in *NSDI*, 2007.
[10] "UMTS," http://www.3gpp.org/technologies/keywords-acronyms/103-umts.
[11] "LTE," http://www.3gpp.org/technologies/keywords-acronyms/98-lte.
[12] "SIP: Session Initiation Protocol," https://www.ietf.org/rfc/rfc3261.txt.
[13] R. Luigi, L. Giuseppe, and M. Vincenzo, "Speeding up packet i/o in virtual machines," in *ANCS*, 2013.
[14] "Intel Data Plane Development Tool Kit," http://dpdk.org/.
[15] S. Vyas, E. Norbert, and R. Sylvia, "Design and implementation of a consolidated middlebox architecture," in *NSDI*, 2012.
[16] M. Nick, A. Tom, and B. Hari, "Openflow: enabling innovation in campus networks," in *SIGCOMM*, 2008.
[17] J. Sushant, K. Alok, and M. Subhasree, "B4: Experience with a globally-deployed software defined wan," in *SIGCOMM*, 2013.
[18] H. Chi-Yao, K. Srikanth, and M. Ratul, "Achieving high utilization with software-driven wan," in *SIGCOMM*, 2013.
[19] R. Yi, P. Tuan, C. Cheng, and Y. Wei, "Dynamic auto scaling algorithm (dasa) for 5g mobile networks," in *GLOBECOM*, 2016.
[20] "ETSI NFV MANO framework," https://wiki.opnfv.org/display/mano/ETSI-MANO.
[21] "IP Location Finder," https://www.iplocation.net/.
[22] S. Brent, C. Alan, and F. Wes, "Past: Scalable ethernet for data centers," in *ACM Proc. of CoNext*, 2012.
[23] "Floodlight," http://www.projectfloodlight.org/floodlight/.
[24] "PJSIP," http://www.pjsip.org/.
[25] "Snort intrusion detection system," https://www.snort.org/.
[26] "SoftLayer," www.softlayer.com.

Jingpu Duan received the B.E. degree from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, in 2013. He is currently pursing the Ph.D degree with the Department of Computer Science, The University of Hong Kong, Hong Kong. His research interests include software defined networking (SDN) and network function virtualization (NFV).

Chuan Wu received her B.E. and M.E. degrees in 2000 and 2002 from Tsinghua University, China,and her Ph.D. degree in 2008 from University of Toronto, Canada. She is currently an associate professor in the Department of Computer Science,The University of Hong Kong. Her research interests include cloud computing and network function virtualisation.

Franck Le is a researcher at IBM T. J. Watson. He received a Ph.D. from Carnegie Mellon University, and a Diplome d'Ingenieur from the Ecole Nationale Superieure des Telecommunications de Bretagne in France.

Alex X. Liu received his Ph.D. degree in Computer Science from The University of Texas at Austin in 2006. He received the IEEE & IFIP William C. Carter Award in 2004, a National Science Foundation CAREER award in 2009, and the Michigan State University Withrow Distinguished Scholar Award in 2011. He is an Associate Editor of IEEE/ACM Transactions on Networking, an Associate Editor of IEEE Transactions on Dependable and Secure Computing, and an Area Editor of Computer Communications. He received Best Paper Awards from ICNP-2012, SRDS-2012, and LISA-2010. His research interests focus on networking and security.

Yanghua Peng received the B.E. degree from the Department of Computer Science, Wuhan University in 2015. He is currently pursuing the Ph.D. degree with the Department of Computer Science, The University of Hong Kong, China. His research interests include data centre networking and resource management.