Simulation Modelling Practice and Theory 120 (2022) 102597

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

CloudSimSFC: Simulating Service Function chains in Multi-Domain Service Networks

Jie Sun [a], Tianyu Wo [b,*], Xudong Liu [a], Rui Cheng [c], Xudong Mou [d], Xiaohui Guo [d], Haibin Cai [e], Rajkumar Buyya [f]

[a] *SKLSDE, School of Computer Science and Engineering, Beihang University, Beijing, China*
[b] *SKLSDE, School of Software, Beihang University, Beijing, China*
[c] *Pinduoduo Corp., Shanghai, China*
[d] *Research Center of Big Data and Computational Intelligence, Hangzhou Institute of Beihang University, Hangzhou, China*
[e] *Software Engineering Institute, East China Normal University, Shanghai, China*
[f] *Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia*

ARTICLE INFO

Keywords:
Multi-domain service network
Service function chain
Deployment strategy
Simulation
Network function virtualization

ABSTRACT

Service Function Chain (SFC) is widely adopted in Multi-domain Service Networks (MDSN) to enforce network policies on customer traffic. Great effort has been devoted to the research of SFC deployment strategies. In this context, simulators are helpful to the design and evaluation of those strategies. A good SFC simulator should be accurate, comprehensive and user-friendly for SFC simulation. However, existing simulators fail to satisfy the requirement due to two major drawbacks: First, they overlook the resource heterogeneity and performance instability of the MDSN environment. Second, their performance models for service functions are too simple to contain the important features such as traffic changing effect and packet queue. To overcome these drawbacks, we propose CloudSimSFC — a new SFC simulator that (1) simulates the performance fluctuations and server failure/recovery events in MDSN environment to align with the performance instability of real-world systems, (2) elaborates the modeling of service functions by incorporating the computation components such as CPU and queue, and the traffic changing effect which happens during packet processing, (3) employs scenario abstraction to simplify the definition of new (heterogeneous) simulation scenarios and supports standard service metrics like request–response time. With these features, CloudSimSFC can be used to simulate SFC run-time performance and evaluate deployment strategies. We introduce the system architecture of CloudSimSFC to explain the simulation principle. We conduct extensive experiments to evaluate the simulation accuracy of CloudSimSFC by comparing it with an SFC prototype. Experimental results confirm that CloudSimSFC achieves 95%+ accuracy in all the evaluation scenarios. We also evaluate the simulation running time and analyze the overhead caused by the proposed simulation features. Finally, we demonstrate the primary usage of CloudSimSFC with case studies.

1. Introduction

Virtualization technology has been widely applied in IT industry to support elastic, on-demand resource provisioning and multi-tenancy. Although proprietary hardware-based network services such as firewalls or load-balancers exhibit good performance, they

* Corresponding author.
E-mail address: woty@act.buaa.edu.cn (T. Wo).

https://doi.org/10.1016/j.simpat.2022.102597
Received 6 December 2021; Received in revised form 30 May 2022; Accepted 31 May 2022
Available online 7 June 2022

fail to cope with the customer's ever-changing resource demand due to their limited support for multi-tenancy and lack of flexibility (mainly caused by their hardware nature). Network Function Virtualization (NFV) and Software Defined Network (SDN) together transform the implementation of network services from proprietary hardware appliances to softwarised Service Functions (SF) running on commodity servers. These two technologies, along with the emerging communication technologies such as 5G have laid the architectural foundation for envisioning an elastic Multi-domain Service Network (MDSN), where new application demands like Industrial Internet of Things (IIoT) monitoring and real-time video conference could be satisfied with performance guarantees.

MDSN can contain multiple computing domains, including clouds and edges, connected through the Wide Area Network (WAN). In MDSN, end-to-end network services are realized through Service Function Chain (SFC). An SFC contains multiple Service Functions (SF) with network- or application-specific purposes. The customer network traffic spans data centers, carrier networks, and edges is steered through the SFC to enforce a series of operations.

SFC promises great service flexibility with the cost of operational complexity. In other words, as the network size grows, the deployment choices would increase exponentially. This problem has lead to an extensive pursue of efficient SFC deployment strategies towards the joint problem of SFC placement and resource allocation. It is however non-trivial to evaluate such strategies through large-scale deployment for the following reasons. First, due to funding limitations, the majority of researchers cannot access large-scale MDSN test-beds. In the meantime, to operate a system at such a scale is always complicated and error-prone. Researchers have to finish the complicated configuration before they could eventually start the evaluation, leading to a severe distraction from the research focus. Besides, reproducing the experiment at scale is difficult because of the unstable performance characters of the resource environment (e.g., server failures, performance fluctuations).

As a result, simulation has been widely used to fill the gap. A moderate-size commodity server can simulate the behaviors of a massive scale network, not to mention that the simulator is easy to control and the experiments are reproducible. With the help of a suitable simulator, researchers could develop new ideas by dedicating their full attention to the core problem rather than wasting time on the pains-taking setup and configuration procedures to operate a physical infrastructure. Innovative ideas could be tested through simulation in prior to be evaluated on physical infrastructure [1].

Many simulators aim for network simulation [2–5] or cloud computing task simulation [6–12]). Although a few simulators support SFC simulation (e.g., CloudSimSDN-NFV [6] and CloudSimNFV [10]), to simulate SFC in MDSN environment is still a challenging task. The challenges remain in the following aspects. First, due to the absence of a high-level resource abstraction, to define a MDSN scenario with existing simulators can be very difficult. In fact, existing simulators use per-item based scenario configuration. Users have to finish complex configurations (e.g., for each switch, port and link) in order to boot a heterogeneous MDSN simulation scenario. Second, an SFC processes network traffic as a series of inter-dependent computation and transmission tasks. The throughput of different types of SFs can vary significantly depending on their processing logic. An SF might also change the traffic volume during processing. However, the computation models in related work are too simple to include these characteristics. Third, users might need to evaluate SFC performance under dynamic environment, since real-world servers could fail and recover, and computation/transmission performance might fluctuate. Nonetheless, existing studies overlook the unstable performance characters of real-world environments.

In this paper, to overcome these drawbacks, we propose CloudSimSFC, a toolkit for simulating SFC in MDSN. The main contributions of this paper are summarized as follows:

- We highlight the requirements for simulating SFC in MDSN environment and propose a simulator named CloudSimSFC. CloudSimSFC can be used to simulate SFC operational performance and evaluate SFC deployment strategies.
- We consider the unstable performance of the underlying resource environment such as performance fluctuations (e.g., SF throughput fluctuations, network jitters) and server failures as observed in large-scale networked systems, enabling CloudSimSFC to reflect realistic scenarios.
- We propose an elaborated SF performance model to involve the distinctive features of SFC. In this model, CPU capacity and packet queue are jointly considered to simulate the SFs. Besides, the traffic changing effect during packet processing is also considered.
- We implement CloudSimSFC by extending an open-source simulator [6]. CloudSimSFC exposes high-level configuration interfaces so that users only need to provide three configuration files to start a simulation. The source code of CloudSimSFC is available at online repository "https://github.com/JasonatBuaa/CloudSimSFC/".
- We evaluate the performance of CloudSimSFC in terms of simulation accuracy and simulation running time. We then demonstrate the primary usage of CloudSimSFC through case studies on typical scenarios.

Roadmap: The rest of the paper is organized as follows. Section 2 presents the background and terminology of this paper; Section 3 presents the overview of CloudSimSFC; Section 4 introduces the simulation model in detail. Section 5 presents the performance experiments and case studies. We discuss the related work in Section 6 and conclude this paper in Section 7.

2. Background

In terms of simulation, there are three relevant roles from SFC market, namely *Resource Provider (RP)*, *SFC customer (customer)*, and *SFC vendor (vendor)*. RPs are the owners of cloud or edge environments who lend their resources to the public on an elastic on-demand basis. Vendors operate SFC systems based on the resources provisioned through multiple RPs, and render SFC services according to customer demands. Customers are the business owners who use SFC to enhance their applications. Vendors always pursue the optimal trade-off between delivering better QoS and minimizing operational expenditure. To this end, CloudSimSFC could help them to evaluate their scheduling policies with simulations.

A Service Function (SF) serves a specific computation purpose like packet filtering or application performance enhancement. Several SFs of different types constitute an SFC to deliver complex functionalities for customers. Consequently, an SFC can be regarded as a series of computation tasks and transmission tasks enforced on the network flows between the ingress (the devices that issue a network request) and egress (the customer application that serves the request) nodes. An SFC policy should include the type and order of SFs, e.g., {Firewall → Network Inspector → L2 forwarder}.

An SFC may span among several heterogeneous domains for latency minimization or cost reduction. Each domain is usually operated by a different RP, and the inter-domain network is typically backed by Wide Area Network (WAN). As for SFC vendors, the domains constitute a large resource pool, referred to as Multi-domain Service Network (MDSN), that can be leveraged as an entirety for resource provisioning and SFC deployment. There are many examples of MDSN in real world. For instance, cloud–edge continuum [13], cloud federation [14], fog computing [15], joint-cloud [16], hybrid cloud [17], etc.

The switches connecting servers and forwarders inside a server along the SFC constitute a packet transmission path. During the SFC deployment process, each SF is mapped to a hosting server (*SFC placement*), and then allocated a resource flavor (*resource allocation*), e.g., [1 core, 2 GB Memory]. Meanwhile, network forwarding rules are installed along the packet transmission path for traffic steering, ensuring that target network traffic should pass through the SFC correctly. The terms *SFC scheduling* and *SFC deployment* are used interchangeably to refer to the joint procedure of *SFC placement* and *resource allocation*.

Inside each SF, many state entries control the computation logic. Each state entry contains at least two fields: A matching condition and the corresponding actions. The matching condition is a hash of the five-tuple with some protocol-specific fields (e.g., TCP ACK, SYN number) or part of the payload data. The incoming network flow is evaluated against the matching conditions on the state entries. If matched, the action on that entry such as allowing, dropping, modifying, or rerouting would be applied (on the network flow).

SFC service latency contains computation latency and transmission latency. When fix the resource flavor, the SFC computation latency is proportional to the size of the workload. As for latency-sensitive applications, if service latency exceeds the deadline, network packets will be dropped, which will in turn damage the customer perceived QoS. Therefore vendors should guarantee the service latency to avoid affecting the performance of customer applications.

3. CloudSimSFC

3.1. Overview

The primary goal of CloudSimSFC is to capture the correlation between SFC scheduling decisions and operational performance through simulation. This goal leads to three principles throughout the design and implementation of CloudSimSFC. First, it is a trade-off between simplify the simulation model for system efficiency and complicate the model for advanced simulation features. With this in mind, we decide to focus on the critical resource characters of both MDSN environments and SF architecture that relate to SFC performance, and simplify other less important aspects. Second, we should also lower the bar on booting a simulation scenario so that users can dedicate their focus to the scheduling policy rather than the simulator configuration. Third, the concepts and structure of CloudSimSFC should be straightforward for our audiences, who are mostly SFC professionals, to understand. CloudSimSFC follows these principles. The modules of CloudSimSFC are shown in Fig. 1.

Scenario Manager maintains the simulation parameters, e.g., to enable/disable the simulation features like server failures or performance fluctuations. Besides, *Scenario Manager* parses the scenario parameters for the *Data Factory* to generate SFC workloads and Failure/Recovery (F/R) events. Specifically, server failures are modeled as F/R events, triggered by the *Discrete-Event Simulator (DES)* during the simulation. We provide the detailed information in Section 3.3.3. *Resource Manager* contains three sub-modules — *Resource Broker* provides the interface for resource provisioning. *Resource Monitor* keeps tracking the allocated and remaining resources. *Deployment Scheduler* provides an interface for users to implement their own scheduling algorithms. When an SFC Demand arrives, the *Deployment Scheduler* parses the parameters and executes the scheduling algorithm to get a feasible result. After that, the *Resource Manager* generates the corresponding events for SFC deployment and resource allocation.

Service Function Perf Module controls the traffic processing operations. As we have addressed before, the traffic processing operations in an SFC can be seen as a series of computation and transmission tasks. These tasks are handled by the *ServiceFunction* and *(Communication) Channel* objects in the simulation environment. In terms of computation task simulation, in addition to the widely adopted CPU performance parameter *Million Instructions Per Second (MIPS)* introduced in [6,7,18], we identify other equally essential contributing factors to SF performance such as *packet queue*, *traffic changing effect (TCE)* and the *performance instability* of the underlying environment.

Discrete-Event Simulator (DES) is the central controller of the event-based simulator. We extend the *DES* of CloudSimSDN-NFV [6] by adding new events and handlers related to the SFC traffic processing operations and the performance instability of the MDSN resource environment.

In a standard simulation process, users are required to define a *simulation scenario* that contains the MDSN resource abstraction (introduced in Section 3.2) and SFC demands (introduced in Section 3.4). *Scenario Manager* parses the simulation scenario to synthesize SFC workloads and F/R events along the simulation timeline. Meanwhile, based on the MDSN resource abstraction, *Scenario Manager* generates the detailed configurations of servers, switches and network links. During the simulation, the *DES* controls the timing for all simulation events, such as request arrival, end of transmission, server failure, etc. The *Deployment Scheduler* processes *SFC Demands* to generate the scheduling policy. The traffic from ingress nodes is steered along the SFC for processing. The service latency is determined by traffic size, task length, link bandwidth, SF capacity, etc. The detailed discussion could be found in Section 4.4.

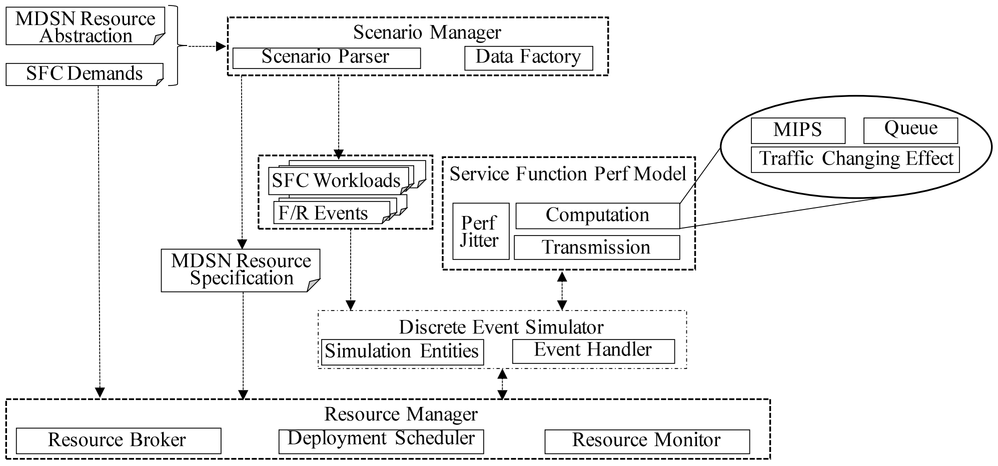


Fig. 1. CloudSimSFC schema.

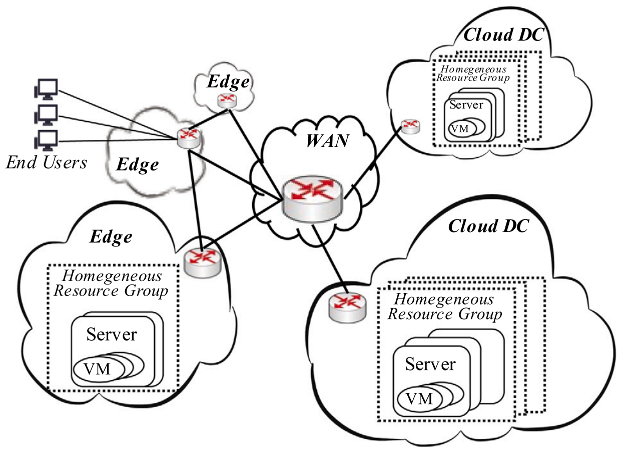

Cloud DC
Homegeneous Resource Group
Server
VM
Edge
WAN
End Users
Edge
Edge
Homegeneous Resource Group
Server
VM
Cloud DC
Homegeneous Resource Group
Server
VM

Fig. 2. The resource abstraction of multi-domain service network.

3.2. Multi-domain service network abstraction

MDSN environment consists of multiple domains with large scale heterogeneous resources. It is infeasible to present MDSN at its full resolution in simulation. On the contrary, CloudSimSFC extracts the critical characters of MDSN that could influence the operation of SFC, into an abstract resource model (i.e., the resource abstraction). This model helps to achieve clarity and simplicity for booting a new simulation scenario.

CloudSimSFC defines *HoMogeneous Resource Group (HMRG)* to refer to the co-located servers with the same characters such as price, flavor and availability, in a domain. Notably, this notation is reasonable since the resource heterogeneity in real-world is observed at cluster level (e.g., rack) rather than server level. By using HMRG as a template of servers, users only need to input server configuration for per HMRG. This could save a lot of effort for defining simulation scenarios (in contrast to other simulators which require per-server configuration).

CloudSimSFC models MDSN into an abstract resource model with layered hierarchy. As is shown in Fig. 2, the outside layer includes the domains and the Wide Area Network (WAN) interconnections. The middle layer corresponds to HMRGs. The innermost layer refers to the servers for hosting SFs. To define a simulation scenario, we only need to consider the domain and HMRG level configuration. The detailed server, switch, link and port configurations could be outsourced to CloudSimSFC.

Domain. There are two types of domains in CloudSimSFC, namely *cloud DC* and *edge*. Cloud DC maintains large scale elastic resources with high availability, whereas edge only contains limited resources to facilitate real-time computation and transmission. Besides, an edge is also the network entry point for the nearby *End Users*. Network traffic (i.e., workloads) is produced from user devices (ingress), and then routed through the SFC to the service endpoint (egress).

Computation Resource. Since the edge environment only possesses limited servers, we simulate each edge with one HMRG. On the other hand, a cloud DC contains thousands of servers with heterogeneous configurations. To be consistent with these

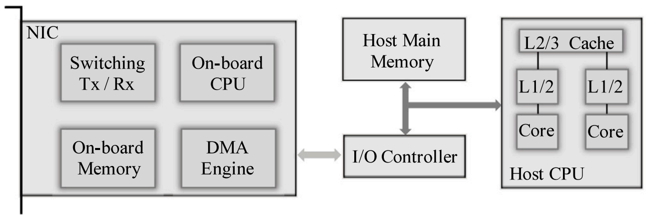


Fig. 3. The CPU and intelligent Network Interface Card plugged on a blade server can be modeled as an intact computation component, containing processing capacity and data caching capacity.

facts, CloudSimSFC allows for multiple HMRGs in a cloud DC. Besides, intelligent Network Interface Cards (iNIC) have been widely deployed in modern computing facilities. The iNIC supports to offload network tasks thanks to the on-board memory and computation units, such that specific tasks like packet queuing or encapsulation/de-encapsulation could be processed efficiently on the iNICs. In CloudSimSFC, we create a uniform computation model that combines the general purpose platform (e.g., blade server) and iNIC (shown in Fig. 3) with packet queuing and processing capacities.

Network. Technologies such as switched-fabric have delivered tremendous performance for local area networks (LAN, e.g., data center network). On the other hand, the inter-domain network connections exhibit limited per-connection bandwidth. In other words, the WAN bandwidth resource is very scarce in contrast to the LAN bandwidth (see e.g., [19,20]). Based on this fact, CloudSimSFC simplifies the network model by only considering the bandwidth limitations for inter-domain networks, while assuming the intra-domain network infrastructure to have unlimited bandwidth. However, since the SFC transmission latency (and payment) are associated with the allocated bandwidth, users are still encouraged to allocate bandwidth for each intra-domain sub-paths. In the simulation settings, the intra-domain bandwidth price should be remarkably lower than that of the inter-domain network by default. Also note that the bandwidth resource of each connection is assumed to be monopolized (rather than shared) by only one corresponding SFC in our model.

3.3. *Performance instability*

Prior simulators (CloudSim, CloudSimSDN-NFV, etc.) denote the CPU capacity with *Million Instructions Per Second (MIPS)* and use *Million Instructions Per Operation (MIPO)* to relate VM/SF throughput with CPU capacity. They only consider the expected performance factors (constant performance, c-perf) while overlooking the widely observed performance fluctuations in shared infrastructure during run-time. However, these fluctuations could lead to a notable difference on the service latency, which have been confirmed by [21]. The performance fluctuations should be addressed to get a more thorough simulation result, especially for latency-critical network services [22]. To this end, CloudSimSFC models both constant and dynamic performance factors for the system performance, denoted as run-time performance (r-perf).

3.3.1. *Constant performance factor (c-perf)*

The packing VM's flavor determines the constant performance (c-perf, represented with million instructions per second) of an SF. Here our basic premises are (1) c-perf is a linear function of flavor. For example, if 1Core/2 GB flavor corresponds to the c-perf of 1000MIPS, then 2Core/4 GB corresponds to 2000MIPS. Based on this idea, a server's processing capacity could be denoted by its c-perf (instead of flavor). (2) the value of c-perf is bounded by server i's remaining capacity at time t, i.e., $c - perf \leq Cap_i(t)$, (3) c-perf is flexible. it can be any positive integer in the feasible region, which is enabled by recent advances in resource-isolation technologies.

3.3.2. *Run-time performance (r-perf)*

A real-world system's run-time performance (r-perf) is affected by many factors, including environmental temperature, system usage, uptime, co-located SF instances, etc. All these factors have led to the observation of performance fluctuations (e.g., SF throughput fluctuations and network jitters) over time.

To simulate the performance fluctuations, CloudSimSFC models r-perf with *Normal Distribution*. To be specific, the MIPS for a server is described with μ and σ^2, where μ corresponds to c-perf while σ^2 defines the extent of performance fluctuations that will influence all the co-located tasks (i.e., SFs) on the server. Likely, the network jitters of intra- and inter-domain networks are also considered. Whenever an SF starts to process a new workload, CloudSimSFC will take a sample from the distribution model as the value of r-perf. This sampling procedure can be performed at per-second/per-workload/per-event fashion throughout the simulation duration.

Notably, although there have been studies on the performance instability in cloud and edge computing, precise modeling of the performance instability is difficult due to multi-tenancy, resource heterogeneity, noisy neighbors, surge in workload and platform (including both hardware and software levels), etc. [23,24]. [25] reports that random disk I/O and network bandwidth can be modeled with normal distribution during the interference of colocated VMs, and [8] also use normal distribution to model the run-time performance of VMs. Therefore, we model the run-time performance with normal distribution in this paper as default. Users can extend the "PerformanceJitter" module of our proposed simulator to replace such default setting with other distributions.

Table 1
An illustration of the Failure/Recovery events.

Event time	Host name	Failure time	Recovery time
0.1	host01	0.2	100.1
121.3	host11	124.1	201.1
121.3	host03	221	231.1
...	...	...	...

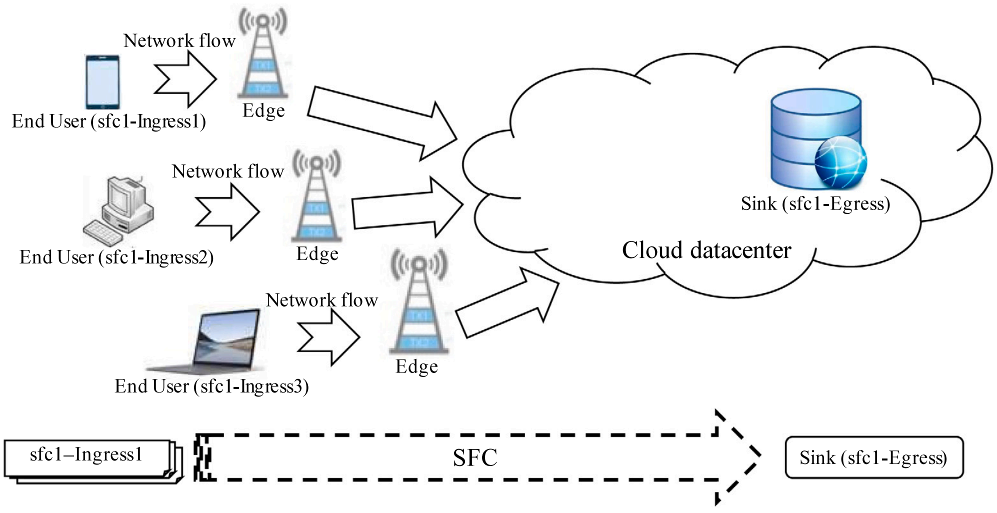


Fig. 4. A logical SFC that contains Ingress nodes from three different edges.

3.3.3. Server failure/recovery

Great efforts have been devoted to pursuing High Availability (HA) to guarantee Service Level Agreement (SLA). Nonetheless, HA is still non-trivial due to the increasing complexity of hardware and software [26,27]. [28,29] have claimed that server failure is one of the top causes of service downtime. If a server fails, all the VMs on it will halt accordingly until the server recovers. To achieve HA, vendors should deploy countermeasures to protect the SFC service from going offline.

CloudSimSFC simulates the failures of physical servers as Failure and Recovery (F/R) events, so that we could evaluate the failure-handling countermeasures through simulation. We use three parameters to control the timing of F/R events — *Server Availability (SA)*, *Mean Time Between Failure (MTBF)* and *Mean Time To Recovery (MTTR)*. Notably, *SA* could be calculated with *MTBF* and *MTTR* according to:

$$SA = \frac{MTBF}{MTBF + MTTR} \tag{1}$$

When defining a simulation scenario, two parameters among *SA*, *MTBF*, and *MTTR* should be provided (per HMRG), then the last parameter is calculated according to Equ (1).

The timing for each F/R event is determined by *SA*, *MTBF*, *MTTR* and the simulation duration T_{sim}. Specifically, CloudSimSFC generates *Next Time To Failure (NTTF)* with exponential distribution and *Next Time To Recovery (NTTR)* with uniform distribution [30] for each server as the default settings, using *MTBF* and *MTTR* respectively as their expectation μ, using T_{sim} to check the validity of the time. Exponential- and uniform- distributions are chosen as default distributions to sample the interval between two F/R events and the duration to recover from a failed status, respectively. In fact, other distributions such as Pareto, Weibull, and Lognormal may also be used here, and it is easy to switch to other distributions in CloudSimSFC.

Examples of F/R events can be found in Table 1. Each F/R event contains “host name”, “event time”, “failure time” and “recovery time”. Notably, the “host name” points to the target server, and the “event time” is left for future usage.

3.4. SFC demand and workload

An SFC demand contains ingress and egress nodes, requested SFC policy, average workload rate (e.g., 1MB/s), and the QoS requirement in terms of latency threshold (if any). Ingress nodes represent the position of user devices (the origination of network requests), which must be placed at an edge domain. On the other hand, egress nodes represent the termination of an SFC, e.g., the customer application that serves the network requests, which could be in any domain. Besides, ingress and egress nodes are dummy nodes that do not consume any resource.

Based on the “average workload rate" parameter in each SFC demand, *Scenario Manager* synthesizes a series of SFC workloads (i.e., network requests) to test the performance of the deployed SFC. The workloads are processed through a series of transmission

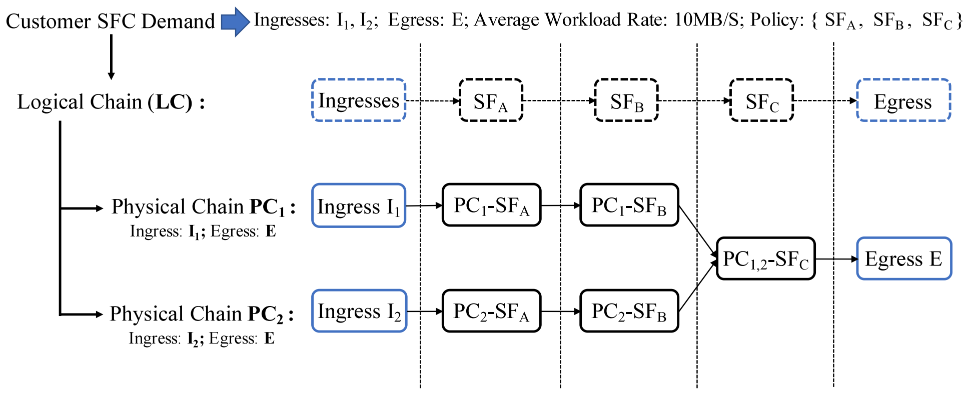


Fig. 5. A logical chain may correspond to multiple physical chains according to the number of Ingress nodes.

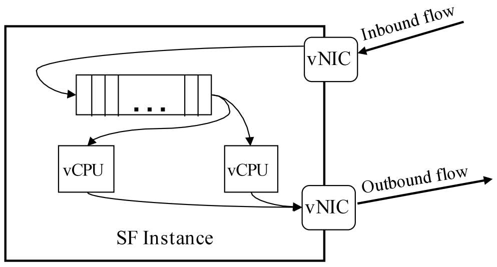


Fig. 6. The computation model of SF.

and computation tasks. As an example, a request to the chain {Ingress →Firewall →Encoder →Decoder → Egress} contains four transmission tasks, i.e., {Ingress →Firewall}, {Firewall →Encoder}, {Encoder →Decoder}, {Decoder →Egress} and three computation tasks on *Firewall*, *Encoder* and *Decoder* respectively.

SFC demands will also be transferred to the *Resource Manager* to invoke the *Deployment Scheduler*, where a deployment strategy will be produced based on the user's algorithm. During this process, the customer demanded SFC is called Logical Chain (LC), and the real SFC deployed between a pair of ingress and egress nodes is called Physical Chain (PC).

Demonstrated in Fig. 4, an LC may contain multiple ingress nodes, indicating that the customer application (i.e., the egress) might be visited by multiple user devices from different edges. On the other hand, a PC only contains one pair of ingress and egress nodes, as is shown in Fig. 5. Besides, SF instances might be shared among PCs based on the deployment strategy of SFC, e.g., "$PC_{1,2} - SF_c$" is shared by "PC_1" and "PC_2".

4. Simulating service function chain

This section introduces the SF performance model of CloudSimSFC. We focus on the distinctive features (regarding system components and operational characters) that influence SFC performance. Before introducing these features, we first briefly introduce the SFC model that we extend from CloudSimSDN-NFV, referred to as the *Basic SFC Model*.

4.1. Basic SFC model

The Basic SFC model follows the *NFV MANO* of European Telecommunications Standards Institute (ETSI). In the basic SFC model, each SF is wrapped into a VM whose resources include CPU (PE) number, CPU speed (MIPS), memory (MB), and bandwidth (MB/s). The computational complexity (MI/MB) is specified for each type of SF, representing the computational cost of processing per unit task (e.g., MB workload). The customer SFC policy is maintained as a *ServiceFunctionChainPolicy (SFCP)*. A component named *ServiceFunctionForwarder (Forwarder)* steers the target network packets into the chained SFs according to the *SFCP* to realize the SFC demand of the customer. Note that during recent years, light-weight virtualization environments such as Docker containers have been widely adopted in production systems. Using a mixture of VMs and containers to host SFs is also a promising solution, e.g., as the hybrid SFC described in [31]. However, as for simulation, CloudSimSFC does not need to distinguish between VMs and containers for hosting SFs. For clarity of expression, we use VM as a general concept to refer to the SF execution environment.

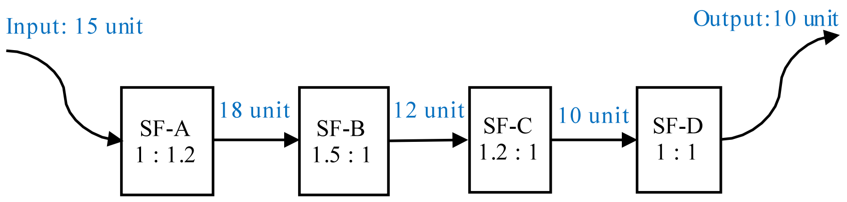


Fig. 7. Illustration of traffic changing effect along an example SFC.

4.2. Packet queue

The throughput of an SF is determined by its computational complexity and resource flavor. The performance is also time-varying and unpredictable due to the dynamic workload, resource contention of co-located tasks, as well as many stochastic factors. As a consequence, different SFs in an SFC usually exhibit inconsistent throughput. Consequently, SF (like other high-throughput networked systems) usually contains a dedicated memory space (i.e, *packet queue*) to cache packets. Hence, SF's computation latency should include both queuing time and processing time. Based on this reason, we introduce packet queue into the SF performance model. As depicted in Fig. 6, incoming traffic will be cached in the packet queue upon arrival to wait for CPU slots to conduct packet processing. Network packets are First-Come-First-Served (FCFS) on each SF by default. To support dedicated scenarios, we can also simulate priority queue that supports preemptive processing.

The queue size is an essential factor for the operation of SFC. Insufficient queue size may lead to packet loss in the case of severe throughput inconsistency, performance fluctuations, or workload burst. To achieve better QoS, CPU and queue size should be jointly considered during the scheduling process. In addition, reactive resource reallocation (e.g., SF scale-up [32], migration [33]) could be applied in case of QoS degradation or packet accumulation (in packet queue).

4.3. Traffic changing effect

The output traffic size of an SF usually deviates from the input size, which has been referred to as the *Traffic Changing Effect* (TCE) [34]. For an SF, the TCE is stable in a long-term view [35,36]. For instance, a firewall performs denying/allowing actions according to the security policy, making the output traffic size less than the input. Citrix WAN optimizer compresses traffic before sending it to the next hop, reducing the traffic volume by 80% [37]. Redundancy eliminator is observed to shrunk the network usage between peak and minimum traffic by 52% [38]. Fig. 7 provides an example of the TCE, where the origin input contains 15 unit traffic and keeps changing along the SFC into the final output of 10 unit traffic. We introduce a parameter named *traffic changing ratio*, i.e., output traffic size to input traffic size, to denote the TCE of an SF. CloudSimSFC changes the output traffic size of an SF based on the traffic changing ratio parameter, e.g., using uniform distribution. Notably, TCE will resize the workload for all subsequent transmission and computation tasks.

It is always rewarding to optimize (i.e., reduce) the inter-domain data transmission volume due to the expensive WAN bandwidth. Fortunately, the TCE of SFs introduces new opportunities for such optimization. TCE could make a significant influence on the SFC deployment decision, which has been confirmed by [34,39–41]. As in the example of Fig. 7, the intuitive idea is to place "SF-A" and "SF-B" in the same domain (if possible).

4.4. Service latency

CloudSimSFC uses *Cloudlet length* to represent the total computation it needs to process a workload and *Million Instruction Per-unit Workload (MIPW)* to represent the computational complexity, i.e., *Cloudlet length* for 1MB workload.

We introduce the underlying mathematics to describe the SFC service latency. Hereafter, capital characters refer to SFC-scope parameters, and lower-case letters refer to SF-scope or link-scope parameters. For any physical SFC S_i, its length is denoted with $|S_i|$ and it contains SFs f^i_j, $j \in [1, |S_i|]$, the type of an SF is given by $type(f^i_j)$. Dummy nodes f^i_0 and $f^i_{|S_i|+1}$ are the endpoints. The link between f^i_k and f^i_{k+1} is l^i_k, $k \in [0, |S_i|]$. For f^i_j, the allocated CPU capacity is c^i_j (MIPS), the computational complexity is p^i_j (MI/MB). For link l^i_k, the allocated bandwidth is b^i_k (MB/s). Denote a workload of S_i as W^i_x, its initial size (on f^i_0 and l^i_0) is w^i_x. When W traverses the SFC, the workload size on link l^i_{j-1} and SF f^i_j is $w^i_{x,j}$. For clearance, we first assume its size conserves during the processing (i.e., Non-TCE), then we get:

$$w^i_{x,j} = w^i_x, j \in [1, |S_i|], \forall i, x \tag{2}$$

Denote the computation time on f^i_j as $tc^i_{x,j}$, which is the sum of queuing time $tq^i_{x,j}$ and workload processing time $tp^i_{x,j}$, i.e.,

$$tc^i_{x,j} = tq^i_{x,j} + tp^i_{x,j} \tag{3}$$

Then tp^i_j could be calculated as:

$$tp^i_{x,j} = (w^i_x \times p^i_j)/c^i_j \tag{4}$$

The transmission time on l^i_j is $tt^i_{x,j}$.

$$tc^i_{x,j} = w^i_x / b^i_j \tag{5}$$

For workload W^i_x, the service time T^i_x on S_i can be calculated with:

$$T^i_x = \sum_{j=1}^{|S_i|} (tc^i_{x,j} + tc^i_{x,j}) \tag{6}$$

Furthermore, we consider the TCE of SFs. Denote the traffic changing ratio of SF f^i_j as r^i_j, then Eq. (2) needs to be replaced by:

$$w^i_{x,j} = w^i_{x,0} \times \prod_{k=1}^{j} r^i_k, j \in [1, |S_i|], \forall i, x \tag{7}$$

where

$$w^i_{x,0} = w^i_x \tag{8}$$

Based on these equations, CloudSimSFC could simulate the per-workload, total or average service time, and SFC's request success rate (or failure rate).

Note that to achieve configuration simplicity, CloudSimSFC assumes the SF computational complexity p^i_j and traffic changing ratio r^i_j are only determined by $type(f^i_j)$. While in a real-world system, the situation might be different. For example, in different SFCs, the deny ratio of a firewall instance often relies on the workload characters and the firewall policy. By this means, users are encouraged to define per SFC parameters when requiring a more accurate model.

5. Performance evaluation and case studies

Having presented the CloudSimSDN architecture, its resource abstraction and the proposed SF model, we now provide a thorough evaluation of the simulator. After this, we then demonstrate the primary usage of CloudSimSFC through case studies. We introduce the simulation scenarios in the case studies and demonstrate the comparison among several scheduling policies in terms of resource consumption and SFC operational performance (i.e., SFC service latency). All the simulations in this section are conducted on a hardware system with an Intel Core i7-9750H CPU (6 cores at 2.6 GHz) and 16 GB Memory (2400 MHz DDR4).

5.1. Accuracy evaluation

5.1.1. Settings

As CloudSimSFC aims to simulate the SFC run-time performance, it is important to output credible, accurate results. To test the simulation accuracy, we implement a prototype that contains a pair of ingress–egress nodes and 5 types of SFs (listed in Table 2). These are typical SFs that are widely used in many researches [34,37,42,43]. We use the prototype output as a baseline.

Prototype. In the prototype, the egress node emulates a file storage server that receives files from clients and logs the request completion time. The ingress node is a client that uploads files to the emulated server. Note that in practice, the length of SFC is usually small (1 to 5) [34,41,44,45]. Hence in this subsection, without lose of generality, we set the chain length to be 3 (shown in Fig. 8). We then deploy the SFC in five VMs of identical configuration. The VMs are provisioned from our private cloud, which uses KVM[1] as hypervisor and serves hundreds of researchers in our LAB. Currently, a vast range of applications are running on the private cloud, including long-term database/web services and short-term scientific computation tasks. We consider these applications as background workloads. We also ensure that in the prototype SFC, each SF uses one dedicated CPU core for workload transmission and processing. The throughput of each SF is tuned to the value listed in Table 2. The computation complexity of each SF is confirmed by related work, e.g., [42,46]. Then the MIPS of each SF can be calculated with Eq. (4). As the computation complexity and MIPS are pure simulation parameters, we denote them as Sim-Complexity and Sim-MIPS respectively in Table 2. We generate a series of workload files with sizes among [5, 20] MB. The bandwidth of each link is specified in Table 3 and controlled by tc[2]. Additionally, we use ntp[3] to synchronize the clock of each SF before running each experiment. We believe this prototype is sufficient to reflect the characteristics of a real world application.

Next, we evaluate the accuracy of CloudSimSFC and CloudSimSDN-NFV. In the first two experiments, we only use Non-TCE SFs (i.e., the SF whose output traffic size equals the input size) for fairness of comparison, since CloudSimSDN-NFV cannot deal with TCE. The evaluated SFC is {Ingress →Firewall →Logger →DPI → Egress}. In experiment 3, we change the target SFC to {Ingress →Firewall →Compressor →Decompressor → Egress}, so as to evaluate CloudSimSFC with TCE SFs. In all the evaluations, we fix the computation/transmission jitter (the σ) used by CloudSimSFC to the same level.

Experiment 1: Non-TCE SFs with steady workload. In this experiment, we generate 1000 consecutive requests at a rate of 1 request/second. We use steady workload: each request uploads a file of size [6, 15] MB with a mean value of 10MB. Hence the

[1] https://www.linux-kvm.org/page/Main_Page
[2] https://man7.org/linux/man-pages/man8/tc.8.html
[3] http://www.ntp.org/documentation.html

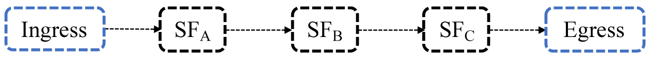


Fig. 8. The prototype SFC to evaluate simulation accuracy. Note that SF$_A$, SF$_B$ and SF$_C$ are chosen from the SFs listed in Table 2. In experiment 1 and 2 of Section 5.1, SF$_A$, SF$_B$ and SF$_C$ are Firewall, Logger and DPI respectively. While in experiment 3, we use Compressor and Decompressor as SF$_B$ and SF$_C$..

Table 2
The prototype SFs and their corresponding simulation parameters.

	Forwarder	Logger	DPI	Compressor	Decompressor
TCR (O/I)	1	1	1	1/3	3/1
Throughput (MB/s)	10	10	10	15	5
Queue (MB)	30	30	30	30	30
Sim-Complexity (MI/MB)	100	150	350	150	450
Sim-MIPS	1000	1500	3500	2250	2250

Table 3
The bandwidth settings for accuracy evaluation.

	Ingress-SF$_A$	SF$_A$-SF$_B$	SF$_B$-SF$_C$	SF$_C$-Egress
Experiment 1	20 MB/s	12.5 MB/s	12.5 MB/s	12.5 MB/s
Experiment 2	20 MB/s	12.5 MB/s	12.5 MB/s	12.5 MB/s
Experiment 3	20 MB/s	16 MB/s	6 MB/s	16 MB/s

Table 4
Simulation accuracy evaluation results. "✗" means the value cannot be acquired in the experiment (since the simulation feature is not supported.).

	Metrics	Prototype	CloudSimSFC	CloudSimSDN-NFV
Experiment 1	Average response time	9.86	9.70	9.48
	Failed request (queue overflow)	80	83	✗
Experiment 2	Average response time	9.91	9.97	14.70
	Failed request (queue overflow)	107	108	✗
Experiment 3	Average response time	7.98	7.64	✗
	Failed request (queue overflow)	94	91	✗

average request rate is approximately 10 MB/s, which equals to the max throughput capacity of the SFC. The bandwidth settings are listed in Table 3. We record the request–response time and the amount of failed requests of the prototype as baseline, and compare them with the simulation results of CloudSimSFC and CloudSimSDN-NFV. We show the results in Fig. 9(a), 9(d) and Table 4.

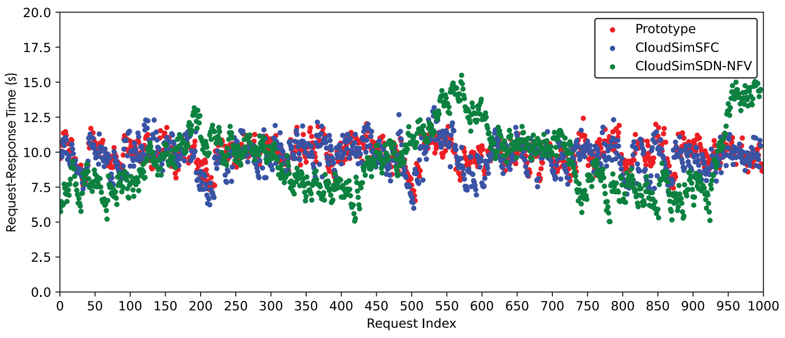
Experiment 2: Non-TCE SFs with workload surge. Based on experiment 1, we further emulate a workload burst by increasing the average request rate by 50% for 60 s starting from the 800-th second. Other settings are the same as experiment 1. The results are shown in Fig. 9(b), 9(e) and Table 4.

Experiment 3: TCE SFs with workload surge. Next, we evaluate the simulation accuracy of CloudSimSFC with TCE SFs. The target SFC is {Ingress →Firewall →Compressor →Decompressor → Egress}. The resource configuration of each SF and link are shown in Tables 2 and 3. The workloads remain the same as in experiment 2. We only report the simulation results of CloudSimSFC, since CloudSimSDN-NFV cannot simulate traffic changing effect. The results are shown in Fig. 9(c), 9(f) and Table 4.

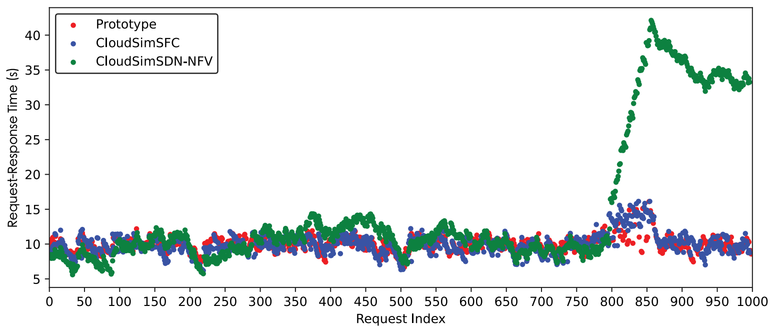
5.1.2. Results

Figs. 9(a), 9(d) and Table 4 indicate that the simulation result of CloudSimSFC in experiment 1 is very close to the baseline. Particularly, the simulation result of CloudSimSFC achieves 98.4% accuracy for average request–response time and 97%, 98.5% and 99.1% accuracy respectively for the quartiles (i.e., the 0.25, 0.5, 0.75 percentile response time). The simulation results of CloudSimSFC are more accurate than CloudSimSDN-NFV, as depicted in Fig. 9(d). This owes to CloudSimSFC's ability to simulate performance fluctuations. To validate this hypothesis, we further conduct a comparative test: we disable CloudSimSFC's performance fluctuation simulation feature and re-conduct experiment 1. In the comparative test, the overall accuracy decreases to the same level as CloudSimSDN-NFV. According to Table 4, we also observe 8% request failure (due to queue overflow) with steady workload in experiment 1, which is quite surprising. We then analyze the logs of the prototype and the simulator for an answer. Consistently, these logs point to the same twofold-reason: First, performance fluctuations increase the queue backlog on each SF and finally lead to high possibility of queue overflow. Second, since our request size ranges from 6MB to 15MB, the 1000 requests form a lot of mini-bursts which could also exhaust the limited queue space.

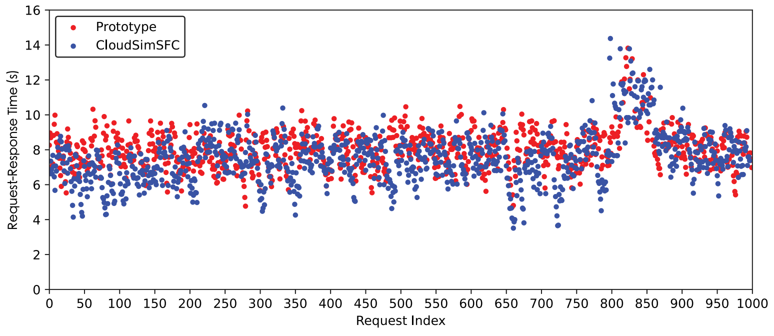
CloudSimSFC's simulation result in experiment 2 remains accurate: 99.4% accuracy for the average request–response time and 99.7%, 99.5% and 98.5% accuracy for the 0.25, 0.5 and 0.75 percentiles respectively. On the other hand, the average response time acquired through CloudSimSDN-NFV is quite accurate when using steady workloads in experiment 1 (97.1%), but the accuracy decreases to 48% as we introduce a 60-second-long workload surge in experiment 2 (see Figs. 9(b), 9(e) and Table 4). The main



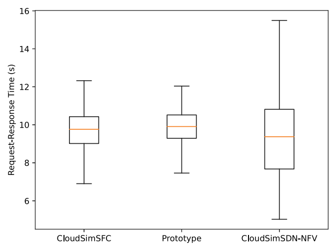
(a) Non-TCE + steady workload



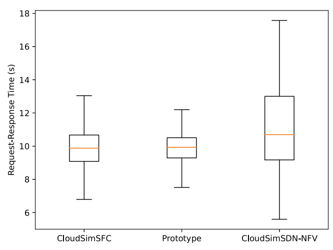
(b) Non-TCE + workload surge



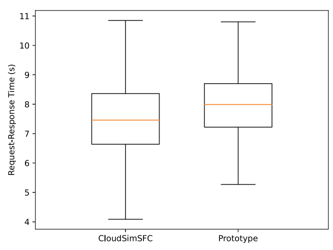
(c) TCE + workload surge



(d) Non-TCE + steady workload



(e) Non-TCE + workload surge



(f) TCE + workload surge

Fig. 9. Simulation accuracy evaluation result. Note that the outliers are not plotted in (d), (e) and (f).

Table 5
Resource scales that we use to evaluate the simulation running time.

	Scale 1 (10 servers)	Scale 2 (22 servers)	Scale 3 (40 servers)
Cloud	2 HMRG * 4 servers	2 HMRG * 8 servers	3 HMRG * 10 servers
Edge	1 HMRG * 2 servers	1 HMRG * 6 servers	1 HMRG * 10 servers

cause is that CloudSimSDN-NFV cannot model the packet queuing behavior. In experiment 2, the average request exceeds the target SFC's max throughput by 50% during the 60-second-long workload surge, hence the SFC queue overflow frequency is higher than that with steady workloads.

As for simulating TCE SFs (experiment 3), Figs. 9(c), 9(f) and Table 4 indicate that CloudSimSFC can achieve 95% accuracy for average request–response time, and the quartiles are still very accurate (96.2%, 96% and 96.4% for the 0.25, 0.5 and 0.75 percentile respectively).

5.2. Simulation running time

CloudSimSFC introduces several simulation features to model the performance of SFC. In fact, more features usually imply higher overhead in terms of event handle and might finally lead to longer simulation running time, especially for discrete event simulators like CloudSimSFC. Next, we measure the simulation running time and discuss the simulation overhead.

5.2.1. Settings

In this experiment, we measure the simulation running time of the proposed simulator. Since our simulator is extended from CloudSimSDN-NFV, we also measure the simulation running time of the latter as a baseline. We construct an evaluation environment that involves an edge and a cloud DC, with three resource scales as shown in Table 5. Additionally, we generate 12 groups of SFC demands by increasing the amount of demand from 5 to 60 with a step size of 5. The SFC length for each demand is chosen from [1,3], and the mean request rate is 10MB/s. For each SFC demand, we generate 1000 requests. Combining the resource scales with the SFC demand groups, we get 36 simulation scenarios for this experiment. We generate all the simulation scenarios and requests by using CloudSimSFC. To ensure accuracy, we run each simulation scenario for ten times and report the average result. The results are depicted in Fig. 10.

Notably, for the sake of fairness, we use the simplified network model as we introduce in Section 3.2 for both CloudSimSFC and CloudSimSDN-NFV. Besides, the traffic changing effect is enabled since it will not cause any overhead.

5.2.2. Results

Simulating performance fluctuation might be expensive, while other features are not. As we simulate performance fluctuation at per-second frequency, Fig. 10 indicates that it prolongs the simulation running time for 26.1%, 26.1% and 27.4% with scale 1, scale 2 and scale 3 respectively (see the lines labeled with CloudSimSFC-Jitter). The reason is quite obvious: CloudSimSFC simulates performance fluctuations with probability distribution models, thus takes a lot of samplings for simulating the computation/transmission tasks. If we simulate performance fluctuations at per-event level, it will require millions of samplings in the experiment with 40 servers and 60 SFC demands. As for other simulation features like queue and FR, the overhead is negligible (9.4%, 2.7% and 1.8% with scale 1, scale 2 and scale 3 respectively) since they barely introduce additional events.

Simulation running time is primarily determined by total SFC demands rather than resource scale. As is depicted in Fig. 10, the simulation running time is always proportional to the amount of SFC demands. As we fix the requests of each SFC demand to 1000, the total simulation events is proportional to the number of SFC demands, hence it takes more simulation running time as the number of demands increases. By contrast, the resource scale only has a marginal impact on simulation running time.

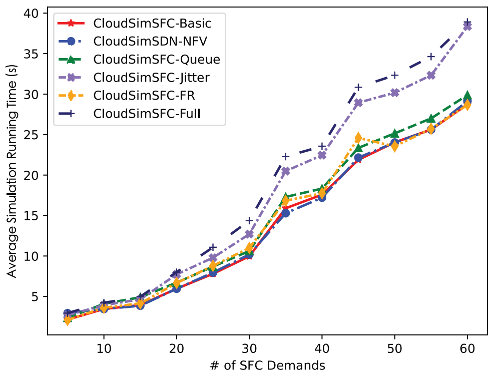
Accuracy or speediness? You can have your own choice. As we have discussed before, simulating performance fluctuations can cause an overhead that prolongs the simulation running time. However, the performance fluctuations are non-negligible since they can make a great influence on system performance [24,47]. This is also echoed by our tests. Therefore, we believe that the overhead is worth taken. We recommend users to use all the proposed simulation features and set the parameters according to their own target physical environment. Again, CloudSimSFC allows users to enable/disable the simulation features through simple configuration.

5.3. Case study 1: Traffic changing effect

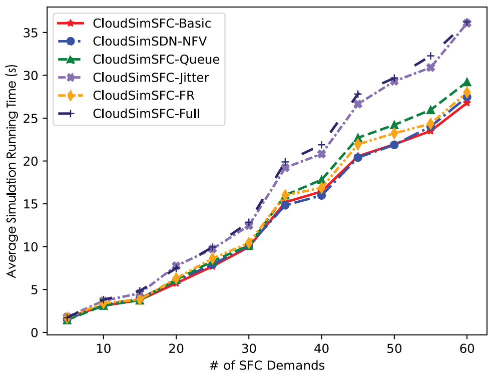
5.3.1. Settings

In this case study, we only focus on the traffic changing effect, so as to achieve clearance of demonstration. Specifically, we disable the packet queue and performance fluctuation simulation features, and set the availability of each server to 100%.

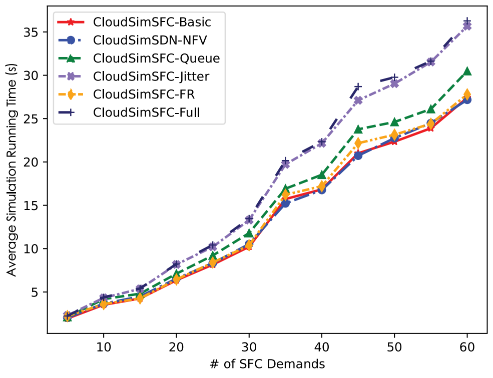
We use a simple MDSN topo to clarify the demonstration. The MDSN topology contains an edge environment and a cloud DC. The total server number is 40 and the detailed configurations of *HMRG* are shown in Fig. 11. We do not limit the bandwidth capacity of the physical links in this case. As for SFC types, we assume the vendors could provide five kinds of SFs with different computational complexities and traffic changing ratios (shown in Table 6). A total of 20 SFC demands are submitted at the beginning of the simulation. The length of each SFC is evenly distributed in $[2, 4]$ and the average request rate of each SFC demand is evenly distributed in $[10, 15]$ MB/s.



(a) Scale 1

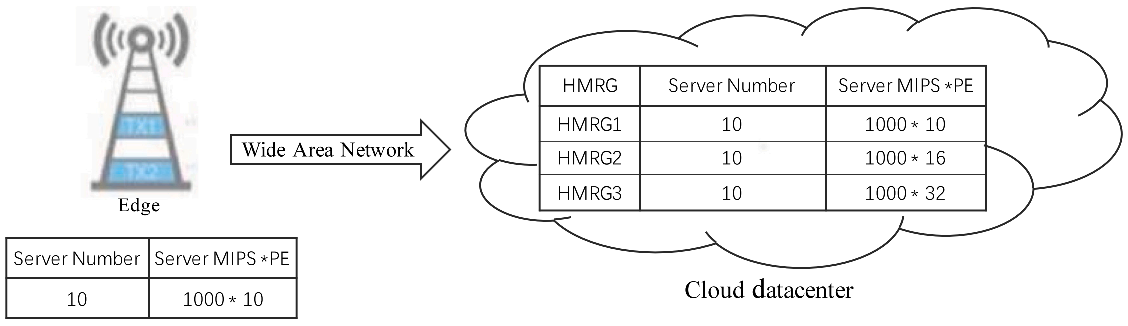


(b) Scale 2



(c) Scale 3

Fig. 10. The simulation running time with 5 to 60 SFC demands with (a) 10 servers, (b) 22 servers and (c) 40 servers in MDSN environment. We evaluate the simulation running time of CloudSimSFC that disables all advanced features (CloudSimSFC-Basic), CloudSimSDN-NFV, CloudSimSFC with queue-aware SF model (CloudSimSFC-Queue), CloudSimSFC with computation performance fluctuations and network jitters (CloudSimSFC-Jitter), CloudSimSFC with F/R events (CloudSimSFC-FR) and the full-fledged CloudSimSFC where all features are enabled(CloudSimSFC-Full).



Server Number	Server MIPS *PE
10	1000 * 10

HMRG	Server Number	Server MIPS *PE
HMRG1	10	1000 * 10
HMRG2	10	1000 * 16
HMRG3	10	1000 * 32

Fig. 11. The MDSN resource configuration of case study 1.

5.3.2. *SFC deployment strategies*

To solve the SFC deployment problem, we divide the deployment process into three stages, namely *SF placement*, *SF resource allocation* and *link bandwidth allocation*. Multiple algorithms are provided for each stage. Note that these algorithms are purely used to demonstrate the capabilities of CloudSimSFC. Thus we do not claim the superiority of the presented algorithms (over others).

- *Stage 1: SF placement.* This stage determines the mappings between each SF and the available domain. We propose for this stage *Traffic Changing Effect-aware MDSN placement algorithm (TCE+MDSN)* and *Cloud Only placement algorithm (CloudOnly)*. The *TCE+MDSN* algorithm leverages TCE to reduce the bandwidth consumption on WAN. On the other hand, *CloudOnly* deploys

Table 6
The SFs used in case study 1. "Expanding" means the output traffic size (of the SF) exceeds the input traffic size, while "Dimmishing" means the opposite.

Type	Computational complexity	Traffic changing ratio (O/I)	Traffic changing pattern
SF-A	50 MI/MB	1.5 : 1.0	Expanding
SF-B	80 MI/MB	1.0 : 2.0	Diminishing
SF-C	150 MI/MB	1.0 : 1.5	Diminishing
SF-D	100 MI/MB	2.0 : 1.0	Expanding
SF-E	200 MI/MB	2.0 : 1.2	Expanding

Table 7
The scheduling policies that we use in case study 1.

Policy index	SF placement	SF resource allocation	Link bandwidth allocation
S1–1	TCE+MDSN	CPX+WI	PerLinkBw
S1–2	TCE+MDSN	CPX+WI	MaxBw
S1–3	TCE+MDSN	CPX+WI	IngressBw
S1–4	TCE+MDSN	NCPX+WI	PerLinkBw
S1–5	CloudOnly	CPX+NWI	PerLinkBw
S1–6	CloudOnly	NCPX+NWI	PerLinkBw

all the SFs in cloud DC, such that workloads must be transmitted to the cloud with their original size. As a consequence, this algorithm may consume more WAN bandwidth.

- *Stage 2: SF resource allocation. Computational comPleXity-aware resource allocation algorithm (CPX)* and *Computational comPleXity agNostic algorithm (NCPX)* are presented. CPX allocates CPU resources according to the computational complexity of SF. On the other hand, NCPX allocates a fixed flavor to each SF regardless of the computational complexity. In addition, we consider the two different conditions according to whether the average workload size for each SF is available, namely *(has) Workload Information (WI)* and *No Workload Information (NWI)*
- *Stage 3: Link bandwidth allocation.* The following policies are considered for bandwidth allocation: *traffic changing effect based Per-Link Bandwidth allocation policy (PerLinkBw)*, *Max-demand based Bandwidth allocation algorithm (MaxBw)*, and *Ingress request rate based Bandwidth allocation algorithm (IngressBw)*. PerLinkBw policy allows to separately allocate bandwidth for each virtual link (i.e., the transmission path from one SF to the next SF in an SFC). MaxBw allocates bandwidth according to the maximum bandwidth requirement of the virtual links in the SFC, while IngressBW allocates bandwidth according to the average request rate of the SFC.

In this case study, we compare the six policies shown in Table 7 for SFC deployment.

5.3.3. Results

Fig. 12 presents the experimental results of case study 1. Fig. 12(a) shows that "S1-1", "S1-2", "S1-3" and "S1-4" use computation resources from across the MDSN environment, while "S1-5" and "S1-6" only use resources from the cloud. The average computation time and average transmission time are shown in Figs. 12(c) and 12(d), which indicate that "S1-1" and "S1-2" achieve better performance in terms of SFC operational efficiency. However, as we can see in Fig. 12(b), policy "S1-2" consumes more WAN bandwidth resources due to the weakness in bandwidth allocation. "S1-6" expresses the worst performance due to insufficient CPU capacity, which can be inferred from the prolonged computation time as indicated in Fig. 12(c). Moreover, by comparing "S1-4", "S1-5" and "S1-6" with "S1-1", we could conclude that knowing only one of the two parameters between *CPX* and *WI* does not necessarily imply a better resource allocation result.

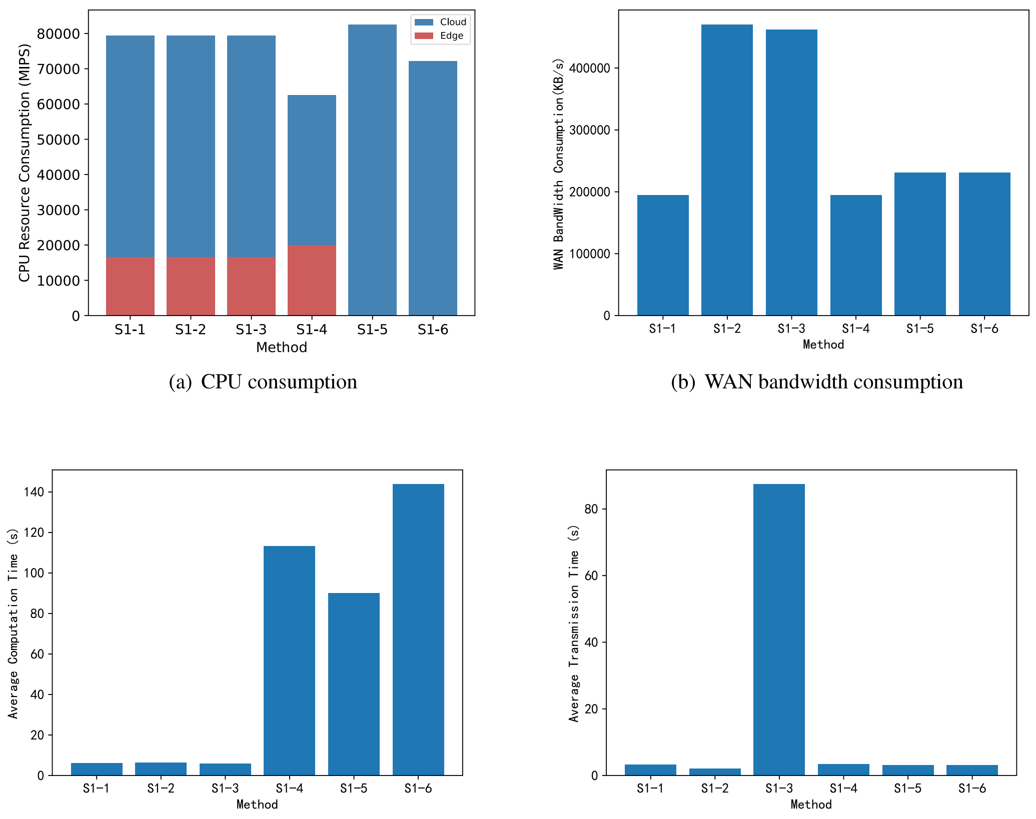
5.4. Case study 2: Performance fluctuations

5.4.1. Settings

Next, we demonstrate the impact of performance fluctuations on SFC operational efficiency. We enable the performance fluctuation simulation feature and vary the level of performance fluctuations to see how they may influence SFC performance. Other experimental settings are the same as in case 1. The details for this case are presented in Table 8.

5.4.2. SFC deployment strategies

In this case study, we use scheduling policy "S1-1" as the baseline since it achieves the best cost-performance trade-off among all the policies in case study 1. In "S2-1", "S2-2", "S2-3" and "S2-4", we allocate resources according to the baseline policy. While in "S2-5", "S2-6", "S2-7" and "S2-8", we allocate 20% more resources to the SF CPU and link bandwidth than the baseline. We vary the level of performance fluctuations (in terms of percentage of c-perf) and plot the results in Fig. 13.



(a) CPU consumption

(b) WAN bandwidth consumption

(c) Average computation time per request

(d) Average transmission time per request

Fig. 12. The experimental results of case study 1. We compare (a) CPU consumption, (b) WAN bandwidth consumption, (c) average computation time and (d) average transmission time among the six policies.

Table 8
The experimental settings of case study 2. "Baseline" refers to the resource allocation policy introduced in "S1-1".

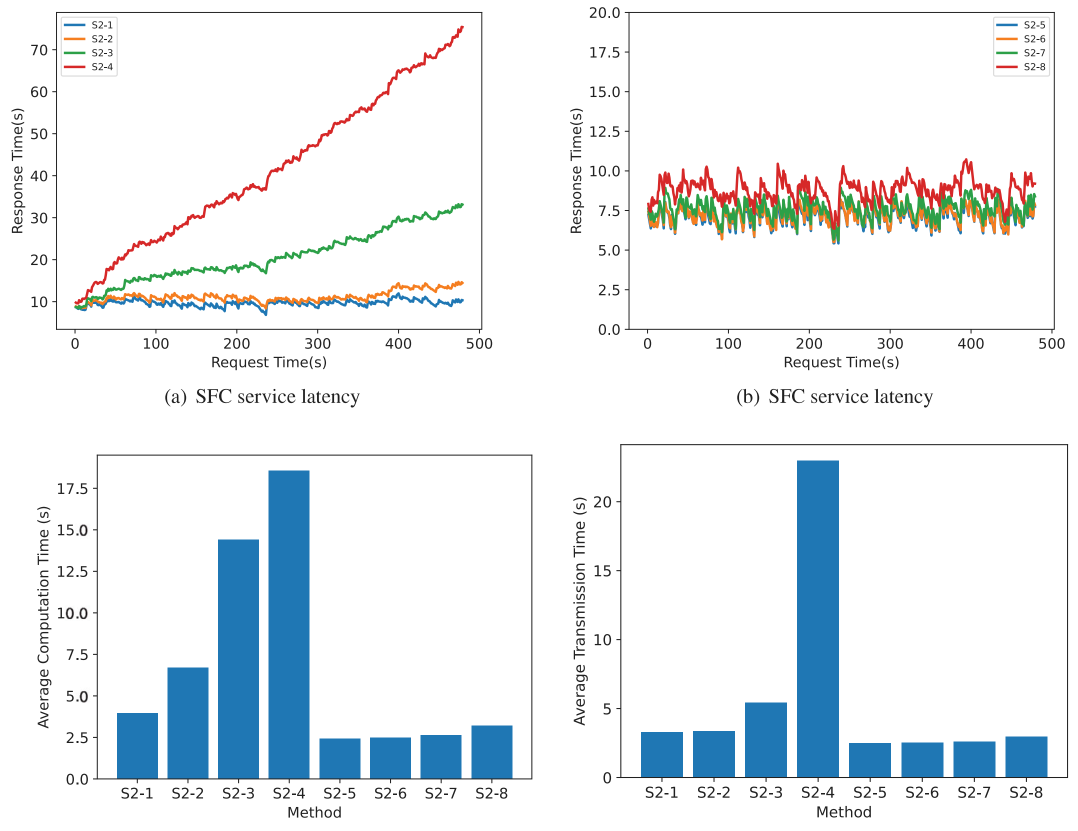
Index	Jitter level	SF resource	Bandwidth allocation
S2–1	1%	Baseline	Baseline
S2–2	5%	Baseline	Baseline
S2–3	10%	Baseline	Baseline
S2–4	20%	Baseline	Baseline
S2–5	1%	1.2 × Baseline	1.2 × Baseline
S2–6	5%	1.2 × Baseline	1.2 × Baseline
S2–7	10%	1.2 × Baseline	1.2 × Baseline
S2–8	20%	1.2 × Baseline	1.2 × Baseline

5.4.3. Results

As is shown in Fig. 13, performance fluctuations have a clear correlation to the request–response time of SFC. Fig. 13(a) shows that the request–response time of the selected SFC is severely affected by the jitter level. As we allocate more resources to the SFC, the impact becomes smaller (see Fig. 13(b)). At the same time, Figs. 13(c) and 13(d) indicate the same result for the overall performance of the 20 SFCs. Based on these findings, we conclude that performance fluctuations must be accounted for to guarantee the SFC QoS when designing scheduling algorithms. Our simulation results suggest to reserve a certain amount of resource for the SFC to tolerate performance fluctuations.

6. Related work

Network Simulators. NS-2 [2] is a discrete event simulator. It is widely used in network research of routing policies, TCP and multicast protocols, etc., over several kinds of wired and wireless (local and satellite) networks. The NS-3 simulator [3] is another network simulator that supports multiple programming languages such as Python and C++. With the provided Application Programming Interfaces(API), NS-3 can execute AI algorithms based on the prevailing AI frameworks such as TensorFlow and



(a) SFC service latency

(b) SFC service latency

(c) Average computation time per request

(d) Average transmission time per request

Fig. 13. The experimental results of case study 2.

Table 9
A comparative analysis of related work including DynamicCloudSim [8], FTCloudSim [9], CloudSimSDN [7], CEPSim [12], CloudSimNFV [10], ACE [11], CloudSimSDN-NFV [6] and CloudSimSFC (Our work).

Simulators	DynamicCloudSim	FTCloudSim	CloudSimSDN	CEPSim	CloudSimNFV	ACE	CloudSimSDN-NFV	CloudSimSFC
First publication	2013	2013	2015	2015	2015	2018	2019	2021 (this paper)
Open source	✓	✓	✓	✗	✗	✓	✓	✓
Performance instability	✓	✗	✗	✗	✗	✗	✗	✓
Server F/R	✗	✓	✗	✗	✗	✓	✗	✓
Workflow model	✓	✗	✓	✓	✓	✓	✓	✓
NFV support	✗	✗	✗	✗	✓	✗	✓	✓
Queue	✗	✗	✗	✗	✗	✗	✗	✓
Traffic changing effect	✗	✗	✗	✗	✗	✗	✗	✓

PyTorch. OMNeT++ [4] is a general network simulation framework that supports to model communication networks, protocols, multiprocessors and distributed hardware systems. GloMoSiM [5] is a library-based simulator aimed purely at wireless network simulation.

Cloud Computing Simulators. CloudSim [18] is an open-source simulator for cloud computing systems that is capable of modeling the workload assignment and execution on a cloud infrastructure. Many simulators [6–12] are developed based on CloudSim. Among them, FTCloudSim [9] models the failure and recovery (F/R) of workloads (i.e., cloudlet) in cloud computing platforms. However, it overlooks the F/R of physical hosts, which is the most common cause of faults in commercial cloud computing platforms. ACE [11] provides an availability-aware extension to CloudSim. It considers HA metrics and failure/redundancy/inter-dependency models to conduct HA-aware simulation. DynamicCloudSim [8] contains a comprehensive performance model which allows simulating various computation tasks with different characteristics (CPU-, I/O-, bandwidth-bound). In addition, it considers the performance instability, including task stragglers and failures, to present a more realistic simulation result. CEPSim [12] extends CloudSim with the ability to simulate the scalability and performance of complex event processing systems. It supports to compare the performance of query processing strategies in multi-domain environments.

SDN and NFV Simulators. CloudSimSDN [7] supports the simulation of SDN-enabled cloud computing environments, including cloud data centers, physical machines, switches and network links. It could be used to tune the scheduling of computation and network resources. CloudSimNFV [10] aims to support NFV scenarios. Meanwhile, CloudSimSDN-NFV [6] shares a similar goal with CloudSimNFV. It facilitates fine-grained SDN control (extended from CloudSimSDN) and load balance among multiple SF replicas. To conclude, we conduct a comparative analysis on related work concerning the following aspects: (1) *basic information* such as the year of publication and the accessibility of source code, (2) *performance instability simulation features* such as performance fluctuations and server failures, (3) *advanced simulation features* such as the modeling for workflow task and NFV (4) *SFC distinctive features* including packet queue and traffic changing effect. Table 9 shows the analysis.

New Trends on Service Function Chains. Recently, a lot of SFC variants emerged. For example, [48,49] proposes an SFC variant named hybrid SFC, where different SFs are used in the forward and backward directions. [50] introduces a variant of SFC where part of the network functions may be provided by dedicated physical hardware while the remainders are provided using virtual instances. Also, a mixture of containers and VMs could be used together to host SF instances, allowing SFC to leverage the collective benefits of different virtualization technologies [31]. Besides, multiple SFs can be executed in parallel when their operations do not conflict, thus providing a better SFC execution model through task parallelization [51]. In this paper, our focus is on the general SFC topology, the advanced features that influence SFC's runtime performance, as well as the MDSN environment resource abstraction. We leave the discussion of these SFC variants for future work.

7. Conclusions and future work

We propose CloudSimSFC, a toolkit for simulating SFC in an MDSN environment. To model the resource features in the MDSN environment, CloudSimSFC considers the heterogeneity and the performance instability of the underlying environment. Besides, CloudSimSFC models the operation of SFC as a series of computation and transmission tasks, and addresses the distinctive features of SFC, such as CPU, packet queue, and traffic changing effect, which are critical for SFC performance simulation. Additionally, CloudSimSFC uses resource abstraction to simplify the definition of new simulation scenarios. Based on these designs, CloudSimSFC is capable of simulating the policies, actions and events regarding the placement and operating of SFCs. We evaluate the simulation accuracy and simulation running time of CloudSimSFC through comparative analysis. Finally, the primary usage of CloudSimSFC is demonstrated through case studies.

Currently, CloudSimSFC uses configuration files to define simulation scenarios. In the future, we will implement a Graphical User Interface (GUI) to simplify the configuration. We also plan to support the new SFC variants in the future.

Software availability: We release CloudSimSFC on repository “https://github.com/JasonatBuaa/CloudSimSFC/".

CRediT authorship contribution statement

Jie Sun: Conceptualization, Investigation, Methodology, Software, Writing – original draft. **Tianyu Wo:** Methodology, Writing – original draft. **Xudong Liu:** Conceptualization, Funding acquisition, Supervision. **Rui Cheng:** Software. **Xudong Mou:** Software. **Xiaohui Guo:** Project administration, Writing – review & editing. **Haibin Cai:** Project administration. **Rajkumar Buyya:** Writing – review & editing, Validation, Supervision.

Acknowledgments

This work is supported in part by the National Key Research and Development Program of China (2018YFB1306000), Ministry of Industry and Information Technology of China (2105-370171-07-02-860873), ShenZhen Science and Technology Innovation Committee, China (CJGJZD20210408092400001) and Melbourne-Chindia Cloud Computing (MC3) Research Network.

References

[1] Jungmin Son, Rajkumar Buyya, Latency-aware virtualized network function provisioning for distributed edge clouds, J. Syst. Softw. 152 (2019) 24–31.
[2] Teerawat Issariyakul, Ekram Hossain, Introduction to network simulator NS2, in: Introduction to Network Simulator NS2, Springer, 2009, pp. 1–18.
[3] George F. Riley, Thomas R. Henderson, The NS-3 network simulator, in: Modeling and Tools for Network Simulation, Springer, 2010, pp. 15–34.
[4] Andras Varga, OMNeT++, in: Modeling and Tools for Network Simulation, Springer, 2010, pp. 35–59, URL https://github.com/omnetpp/omnetpp.
[5] Xiang Zeng, Rajive Bagrodia, Mario Gerla, GloMoSim: A library for parallel simulation of large-scale wireless networks, in: Proceedings. Twelfth Workshop on Parallel and Distributed Simulation PADS'98 (Cat. No. 98TB100233), IEEE, 1998, pp. 154–161.
[6] Jungmin Son, Tian Zhang He, Rajkumar Buyya, CloudSimSDN-NFV: Modeling and simulation of network function virtualization and service function chaining in edge computing environments, Softw. Pract. Exp. 49 (6) (2019) 1748–1764.
[7] Jungmin Son, Amir Vahid Dastjerdi, Rodrigo N Calheiros, Xiaohui Ji, Young Yoon, Rajkumar Buyya, CloudsimSDN: Modeling and simulation of software-defined cloud data centers, in: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE, 2015, pp. 475–484.
[8] Marc Bux, Ulf Leser, DynamicCloudSim: Simulating heterogeneity in computational clouds, Future Gener. Comput. Syst. 46 (2015) 85–99.
[9] Zhou Ao, Shang-Guang Wang, Qibo Sun, Zou Hua, Fangchun Yang, FTCloudSim: A simulation tool for cloud service reliability enhancement mechanisms, in: Proceedings Demo & Poster Track of ACM/IFIP/USENIX International Middleware Conference, 2013.
[10] Wutong Yang, Minxian Xu, Guozhong Li, Wenhong Tian, CloudSimNFV: Modeling and simulation of energy-efficient NFV in cloud data centers, 2015, arXiv preprint arXiv:1509.05875.
[11] Manar Jammal, Hassan Hawilo, Ali Kanso, Abdallah Shami, ACE: Availability-aware CloudSim extension, IEEE Trans. Netw. Serv. Manag. 15 (4) (2018) 1586–1599.
[12] Wilson A. Higashino, Miriam A.M. Capretz, Luiz F. Bittencourt, CEPSim: Modeling and simulation of complex event processing systems in cloud environments, Future Gener. Comput. Syst. 65 (2016) 122–139.

[13] Kaihua Fu, Wei Zhang, Quan Chen, Deze Zeng, Minyi Guo, Adaptive resource efficient microservice deployment in cloud-edge continuum, IEEE Trans. Parallel Distrib. Syst. 33 (8) (2022) 1825–1840, http://dx.doi.org/10.1109/TPDS.2021.3128037.
[14] Tobias Kurze, Markus Klems, David Bermbach, Alexander Lenk, Stefan Tai, Marcel Kunze, Cloud federation, Cloud Comput. 2011 (2011) 32–38.
[15] Shanhe Yi, Zijiang Hao, Zhengrui Qin, Qun Li, Fog computing: Platform and applications, in: 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), 2015, pp. 73–78, http://dx.doi.org/10.1109/HotWeb.2015.22.
[16] Huaimin Wang, Peichang Shi, Yiming Zhang, Jointcloud: A cross-cloud cooperation architecture for integrated internet service customization, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 2017, pp. 1846–1855, http://dx.doi.org/10.1109/ICDCS.2017.237.
[17] Aishwarya Srinivasan, Md Abdul Quadir, V. Vijayakumar, Era of cloud computing: A new insight to hybrid cloud, Procedia Comput. Sci. 50 (2015) 42–51, http://dx.doi.org/10.1016/j.procs.2015.04.059, Big Data, Cloud and Computing Challenges.
[18] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, Rajkumar Buyya, CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. Pract. Exp. 41 (1) (2011) 23–50.
[19] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S. Pai, Michael J. Freedman, Aggregation and degradation in jetstream: Streaming analytics in the wide area, in: 11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14), 2014, pp. 275–288.
[20] Ashish Vulimiri, Carlo Curino, P. Brighten Godfrey, Thomas Jungblut, Jitu Padhye, George Varghese, Global analytics in the face of bandwidth and regulatory constraints, in: 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15), 2015, pp. 323–336.
[21] István Pelle, János Czentye, János Dóka, Balázs Sonkoly, Towards latency sensitive cloud native applications: A performance study on aws, in: 2019 IEEE 12th International Conference on Cloud Computing, CLOUD, IEEE, 2019, pp. 272–280.
[22] Charalampos Stylianopoulos, Magnus Almgren, Olaf Landsiedel, Marina Papatriantafilou, Trevor Neish, Linus Gillander, Bengt Johansson, Staffan Bonnier, On the performance of commodity hardware for low latency and low jitter packet processing, in: Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems, 2020, pp. 177–182.
[23] Shin-gyu Kim, Hyeonsang Eom, Heon Y. Yeom, Virtual machine consolidation based on interference modeling, J. Supercomput. 66 (3) (2013) 1489–1506.
[24] Philipp Leitner, Jürgen Cito, Patterns in the Chaos—A study of performance variation and predictability in public IaaS clouds, ACM Trans. Internet Technol. 16 (3) (2016) http://dx.doi.org/10.1145/2885497.
[25] Jörg Schad, Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Runtime measurements in the cloud: Observing, analyzing, and reducing variance, Proc. VLDB Endow. 3 (1–2) (2010) 460–471, http://dx.doi.org/10.14778/1920841.1920902.
[26] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, Kurnia J. Eliazar, Why does the cloud stop computing? Lessons from hundreds of service outages, in: Proceedings of the Seventh ACM Symposium on Cloud Computing, 2016, pp. 1–16.
[27] M. Fatimah, S. Saad, An analytical model for availability evaluation of cloud service provisioning system, Int. J. Adv. Comput. Sci. Appl. 8 (6) (2017) 240–247.
[28] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, et al., Predicting node failure in cloud service systems, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018, pp. 480–490.
[29] Kashi Venkatesh Vishwanath, Nachiappan Nagappan, Characterizing cloud computing hardware reliability, in: Proceedings of the 1st ACM Symposium on Cloud Computing, 2010, pp. 193–204.
[30] Praveen Yalagandula, Suman Nath, Haifeng Yu, Phillip B. Gibbons, Srinivasan Seshan, Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems., in: First Workshop on Real, Large Distributed Systems (WORLDS 04), USENIX Association, 2004.
[31] Francisco Carpio, Wolfgang Bziuk, Admela Jukan, On optimal placement of hybrid service function chains (SFCs) of virtual machines and containers in a generic edge-cloud continuum, 2020, CoRR abs/2007.04151 arXiv:2007.04151.
[32] Dong Zhai, Xiangru Meng, Zhenhua Yu, Hang Hu, Xiaoyang Han, A fine-grained and dynamic scaling method for service function chains, Knowl.-Based Syst. 228 (2021) 107289.
[33] TianZhang He, Adel N. Toosi, Rajkumar Buyya, SLA-aware multiple migration planning and scheduling in SDN-NFV-enabled clouds, J. Syst. Softw. 176 (2021) 110943.
[34] Wenrui Ma, Oscar Sandoval, Jonathan Beltran, Deng Pan, Niki Pissinou, Traffic aware placement of interdependent NFV middleboxes, in: IEEE INFOCOM 2017-IEEE Conference on Computer Communications, IEEE, 2017, pp. 1–9.
[35] Aaron Gember, Anand Krishnamurthy, Saul St John, Robert Grandl, Xiaoyang Gao, Ashok Anand, Theophilus Benson, Vyas Sekar, Aditya Akella, Stratos: A network-aware orchestration layer for virtual middleboxes in clouds, 2013, arXiv preprint arXiv:1305.0209.
[36] Yifu Yao, Songtao Guo, Pan Li, Guiyan Liu, Yue Zeng, Forecasting assisted VNF scaling in NFV-enabled networks, Comput. Netw. 168 (2020) 107040.
[37] Critix, Citrix WAN Optimization With NetScaler SD-WAN URL https://www.citrix.com/content/.
[38] Ashok Anand, Chitra Muthukrishnan, Aditya Akella, Ramachandran Ramjee, Redundancy in network traffic: Findings and implications, in: Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems, 2009, pp. 37–48.
[39] Guangxiong Wu, Feilong Tang, Lijun Cao, Ping Han, Yanqin Yang, Wenchao Xu, Xingjun Zhang, Dynamic processing while transmitting for SDN-based space-terrestrial integrated networks, in: 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), IEEE, 2020, pp. 499–506.
[40] Zhe Wang, Zhiwei Zhao, Chang Shu, Geyong Min, Yunpeng Han, Yuhong Jiang, Orchestrating service function chains with joint resource optimization in NFV networks, in: 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), IEEE, 2019, pp. 1115–1122.
[41] Yang Chen, Jie Wu, NFV middlebox placement with balanced set-up cost and bandwidth consumption, in: Proceedings of the 47th International Conference on Parallel Processing, 2018, pp. 1–10.
[42] Amin Tootoonchian, Aurojit Panda, Chang Lan, Melvin Walls, Katerina Argyraki, Sylvia Ratnasamy, Scott Shenker, Resq: Enabling SLOs in network function virtualization, in: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX Association, Renton, WA, 2018, pp. 283–297, URL https://www.usenix.org/conference/nsdi18/presentation/tootoonchian.
[43] Behrooz Farkiani, Bahador Bakhshi, S. Ali MirHassani, Tim Wauters, Bruno Volckaert, Filip De Turck, Prioritized deployment of dynamic service function chains, IEEE/ACM Trans. Netw. 29 (3) (2021) 979–993, http://dx.doi.org/10.1109/TNET.2021.3055074.
[44] A. Tomassilli, F. Giroire, N. Huin, S. Pérennes, Provably efficient algorithms for placement of service function chains with ordering constraints, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 774–782, http://dx.doi.org/10.1109/INFOCOM.2018.8486275.
[45] Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba, Otto Carlos Muniz Bandeira Duarte, Orchestrating virtualized network functions, IEEE Trans. Netw. Serv. Manag. 13 (4) (2016) 725–739, http://dx.doi.org/10.1109/TNSM.2016.2569020.
[46] Bhavish Aggarwal, Aditya Akella, Ashok Anand, Athula Balachandran, Pushkar Chitnis, Chitra Muthukrishnan, Ramachandran Ramjee, George Varghese, EndRE: An end-system redundancy elimination service for enterprises, in: 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10), USENIX Association, San Jose, CA, 2010, URL https://www.usenix.org/conference/nsdi10-0/endre-end-system-redundancy-elimination-service-enterprises.
[47] Jeanna Neefe Matthews, Wenjin Hu, Madhujith Hapuarachchi, Todd Deshane, Demetrios Dimatos, Gary Hamilton, Michael McCabe, James Owens, Quantifying the performance isolation properties of virtualization systems, in: Proceedings of the 2007 Workshop on Experimental Computer Science, 2007, pp. 6–es.

[48] Danyang Zheng, Chengzong Peng, Xueting Liao, Ling Tian, Guangchun Luo, Xiaojun Cao, Towards latency optimization in hybrid service function chain composition and embedding, in: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, 2020, pp. 1539–1548, http://dx.doi.org/10.1109/INFOCOM41043.2020.9155529.
[49] Danyang Zheng, Chengzong Peng, Xueting Liao, Xiaojun Cao, Toward optimal hybrid service function chain embedding in multiaccess edge computing, IEEE Internet Things J. 7 (7) (2020) 6035–6045, http://dx.doi.org/10.1109/JIOT.2019.2957961.
[50] Hendrik Moens, Filip De Turck, Customizable function chains: Managing service chain variability in hybrid NFV networks, IEEE Trans. Netw. Serv. Manag. 13 (4) (2016) 711–724, http://dx.doi.org/10.1109/TNSM.2016.2580668.
[51] Danyang Zheng, Gangxiang Shen, Xiaojun Cao, Biswanath Mukherjee, Towards optimal parallelism-aware service chaining and embedding, IEEE Trans. Netw. Serv. Manag. (2022) 1, http://dx.doi.org/10.1109/TNSM.2022.3142184.

Jie Sun is a Ph.D candidate in the State Key Laboratory of Software Development Environment (SKLSDE), School of Computer Science and Engineering, Beihang University, Beijing, China. His current research interests lies in cloud computing, software defined networking and network virtualization. He has been actively involved in National R&D Programs regarding cloud computing and virtualization, with a special focus on hybrid edge–cloud cooperative computing technologies. He obtained B.S. and M.S. degrees in College of Information Science and Engineering, Ocean University of China, in 2012 and 2015 respectively.

Tianyu Wo is an Associate Professor with State Key Laboratory of Software Development Environment (SKLSDE), School of Computer Science and Engineering, Beihang University, Beijing, China. His main research directions include distributed systems and industrial Internet software technology.

Xudong Liu is a Professor with State Key Laboratory of Software Development Environment (SKLSDE), School of Computer Science and Engineering, Beihang University, Beijing, China. He serves as an Executive Director of W3C China Division. His research interests include software development methods, trusted middle-ware, and networked systems. He is the principal investigator/research team leader of several national key R&D projects in China.

Rui Cheng is a senior software engineer with Pinduoduo Corp. His research focuses on cloud computing and service-oriented computing. He obtained the B.S. degree in College of Computer Science and Technology, Anhui University of Technology, and M.S degree in School of Computer Science and Engineering, Beihang University.

Xudong Mou is a Researcher with the Research Center of Big Data and Computational Intelligence, Hangzhou Institute of Beihang University. She received the B.S. and M.S. degrees in School of Computer Science and Engineering, Beihang University, in 2017 and 2021, respectively. Her current research interests lie in industrial software and virtual network technologies.

Xiaohui Guo is a Researcher in the Research Center of Big Data and Computational Intelligence, Hangzhou Innovation Institute, Beihang University. He received his Ph.D. degree from School of Computer Science and Engineering, Beihang University, Beijing, China. His research interests include artificial intelligence, cloud–edge collaborative intelligent services, cloud robotics systems.

Haibin Cai is a professor in the Software Engineering Institute, East China Normal University. He received his Ph.D. degree from the Donghua University in 2008, Shanghai, China. He received his B.Eng. from the National University of Defense Technology in 1997, and M.S. degree from the National University of Defense Technology in 2004. His research interests include trusted artificial intelligence, intelligent perception and trustworthy computing.

Dr. Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in cloud computing. He has authored over 850 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. Software technologies for grid and cloud computing developed under Dr. Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 50+ countries around the world. He served as the founding Editor-in-Chief of the IEEE Transactions on Cloud Computing. He is currently serving as Co-Editor-in-Chief of Journal of Software: Practice and Experience, which was established 50+ years ago. For further information on Dr. Buyya, please visit his cyberhome: https://www.buyya.com.