Journal of Systems Architecture 116 (2021) 101956

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

IoTSim-Osmosis: A framework for modeling and simulating IoT applications over an edge-cloud continuum

Khaled Alwasel [a,b,*], Devki Nandan Jha [a], Fawzy Habeeb [a], Umit Demirbaga [a,c], Omer Rana [d], Thar Baker [e], Scharam Dustdar [f], Massimo Villari [g], Philip James [a], Ellis Solaiman [a], Rajiv Ranjan [a,h]

[a] Newcastle University, Newcastle upon Tyne, UK
[b] Saudi Electronic University, Riyadh, Saudi Arabia
[c] Bartin University, Bartin, Turkey
[d] Cardiff University, Cardiff, UK
[e] University of Sharjah, United Arab Emirates
[f] TU Wien, Vienna, Austria
[g] University of Messina, Italy
[h] Chinese University of Geosciences, Wuhan, China

ARTICLE INFO

Keywords:
Osmosis computing
Internet of Things (IoT)
Edge computing
Cloud computing
Software-Defined Network (SDN)
Simulation.

ABSTRACT

The osmotic computing paradigm sets out the principles and algorithms for simplifying the deployment of Internet of Things (IoT) applications in integrated edge-cloud environments. Various existing simulation frameworks can be used to support integration of cloud and edge computing environments. However, none of these can directly support an osmotic computing environment due to the complexity of IoT applications and heterogeneity of integrated edge-cloud environments. Osmotic computing suggests the migration of workload to/from a cloud data center to edge devices, based on performance and security *trigger* events. We propose 'IoTSim-Osmosis– a simulation framework to support the testing and validation of osmotic computing applications. In particular, our detailed related work analysis demonstrates that IoTSim-Osmosis is the first simulation framework to enable unified modeling and simulation of complex IoT applications over heterogeneous edge-cloud environments. IoTSim-Osmosis is demonstrated using an electricity management and billing application case study, for benchmarking various run-time QoS parameters, such as IoT battery use, execution time, network transmission time and consumed energy.

1. Introduction

The Internet of Things (IoT) infrastructure is now used across a number of applications, such as smart city, healthcare, and manufacturing. In such applications, data from IoT devices can be processed by different resources at edge and cloud data centers [1]. The transition from distributed systems (e.g. cloud computing) to a distributed *system of systems* – with the edge and cloud acting as independently managed systems to support an IoT ecosystem, has led to the new generation of heterogeneous and complex environments. Although the platforms that support a *system of systems* perspective may vary, a common theme is to link different types of distributed systems in a unified manner. Osmotic computing focuses on the design and implementation of a unified computing model that leverages the capabilities of various distributed systems, which include edge computing, cloud computing and a software-defined wide area network (SD-WAN) [2]. The aim is to optimize the performance of the overall IoT ecosystem as well as the performance of individual components that are part of such an ecosystem.

Consider an example of electricity management and billing in a smart city. Each house has a smart meter installed to capture electricity consumption on a per-second basis, with a local display showing live meter reading and monetary cost. Periodically (usually at 15 min), the reading is sent to a central server for combined billing, load management and outage control. This process seems simple, and yet also extremely complex, as every smart meter can have a different architecture and follow different communication protocols based on network range and availability. Local processing and transmission in a smart meter may also vary based on its charging state, processing and storage capability. The data of every smart meter enroute to cloud

* Corresponding author at: Newcastle University, Newcastle upon Tyne, UK.
E-mail address: kalwasel@gmail.com (K. Alwasel).

https://doi.org/10.1016/j.sysarc.2020.101956
Received 29 July 2020; Received in revised form 15 November 2020; Accepted 23 November 2020
Available online 28 November 2020

datacenters may experience network congestion, latency and buffering, and use security controls (e.g., firewalls, encryption).

Considering customized solutions for Osmotic computing in a production environment can be challenging. Various research projects including CHOReOS [3], CityPulse [3] are proposed to investigate the infrastructure composition and data analytics operations. The main aim of these projects is to orchestrate numerous services to offer a unified deployment solution. However, they are not involved in the analysis of proposed approaches or algorithms for the deployment. Therefore, simulation-based tools are a useful alternative for analyzing and evaluating algorithms and Quality of Service (QoS) policies — to undertake various "what if" investigations. There are several simulation environments currently available such as IoTSim-Edge [4], IoTsuite [5], SimIoT [6], amongst others for IoT systems. However, their key focus remains on IoT systems and devices, and therefore they have limited capability to model migration and dynamic container management required in osmotic computing within a unified simulation model.

This paper describes IoTSim-Osmosis— an SDN-based osmotic computing toolkit. IoTSim-Osmosis supports modeling and simulation of multiple Osmotic systems in a unified environment. It enables the integration of IoT, edge and cloud ecosystems along with mechanisms to support SD-WAN networking. Using this toolkit, IoT devices are able to send data using different wireless technologies (e.g., WiFi,) while the edge can include virtualized devices and SDN-aware infrastructure. Similarly, a cloud data center can include virtualized host machines and SDN-aware networks. IoTSim-Osmosis also provides policies to control different components (e.g. edge and cloud task scheduling, and edge to cloud routing protocols).

1.1. Challenges

Modeling and simulating IoT-based osmotic environments present the following challenges:

- **Infrastructure heterogeneity:** Osmotic computing is based on the use of a multi-layered architecture (comprising IoT, edge, cloud and SD-WAN), which requires coordination between the layers. Each layer is constantly evolving with heterogeneous components, data formats, and protocols, which might involve a number of different behaviors and configuration parameters (e.g. type of power source, processing/storage capacity, network capacity etc.) [7].
- **Communication heterogeneity** In osmotic environments, several message exchange formats may co-exist between different devices. The increasing use of software-defined networking (SDN) in edge and cloud environments adds an additional potential layer of configuration [8]. The use of SD-WAN in osmotic environments imposes several challenges, such as obtaining a guaranteed network QoS and the requirement of reserving network slices [9].
- **Complexity of application graph:** Increasingly, IoT applications can be represented as a graph of microservices, with data and control flow dependencies between such services encoded in the graph. This use of microservices enables re-use and a mechanism to integrate services offered by a variety of different providers. Each application componentservice in the graph can have specific functional and QoS requirements for successful execution of the application [10].

1.2. Contributions

This paper describes a novel framework to model and simulate osmotic computing environments, based on the characteristics outlined above. Our key contributions include:

- The architecture and systems model of IoTSim-Osmosis, highlighting components used to support edge-cloud heterogeneity and IoT application complexity. This architecture also includes several system management policies that can be extended by other researchers.
- A case study based validation of IoTSim-Osmosis using an energy (electricity) management and billing application. Simulation results highlight the unique capabilities provided by IoTSim-Osmosis for analyzing various parameters, such as IoT energy usage, execution time and network transmission delay.

The rest of the paper is organized as follows. Section 2 describes osmotic computing and graph-based IoT application construction. Section 3 discusses the modeling capabilities of IoTSim-Osmosis and Section 4 provides an empirical validation of our approach. Before providing concluding comments and future work in Section 6, we describe related work in Section 5 by comparing IoTSim-Osmosis with state-of-art efforts.

2. Background

2.1. IoT environment

Although actual IoT infrastructure can vary across different application areas, a common (abstract) model can be represented using a 4-tier architecture as shown in Fig. 1. The four tiers are:

Tier 1. *IoT layer:* This layer can consist of sensors, actuators, Radio Frequency IDentification (RFID) tags and mobile devices, which can sense the physical environment and transfer data to edge/cloud data centers for further analysis [11,12]. These devices can consist of different software/hardware (and data usage) architecture, energy sources and communication protocols. Unlike devices and networks which exist to offer physical connectivity, network-connected applications create opportunities for human-to-device connectivity [13].

Tier 2. *Edge layer:* For applications with the following properties: (a) close coupling between data generators and processing environments [14], (b) where data transfer bandwidth is limited [15], and (c) data generating devices are battery operated [1], it is not efficient to send all the data to a cloud system. Emergence of edge computing which offers data storage and analysis to the network edge closer to IoT devices provides an efficient solution. Edge devices, including smart phone, Raspberry Pi and UDOO board, favor local processing and data storage in proximity to data generation. Similar to IoT devices, edge devices can be heterogeneous, which makes the modeling complex.

Tier 3. *Network layer:* This layer is involved in transferring data between various IoT infrastructure layers. The sensor and actuator nodes (Tier 1) form arbitrary network topologies that are interconnected via edge gateways (Tier 2) to remote clouds (Tier 4) via the Internet backbone. The inter-connectivity of these network types vary from short-range low-power wireless links offering a bandwidth of few hundred kb/s with a radio range of few meters, to powerful local and cellular area networks. There is often direct communication between an IoT device and an edge device using light weight network protocols such as LoRa-WAN, NB-IoT (over long distances) and Bluetooth Low Energy (over shorter distances), whereas edge and cloud layers use network protocols such as 4G/5G [16]. The dynamic nature of modern IoT applications requires dynamic reconfiguration of network links and support for bandwidth slicing, which requires a move away from traditional WAN solutions towards SD-WAN solutions [17].

Tier 4. *Cloud layer:* This layer provides computing as a utility service which can be provisioned on a pay-per-use basis, as user demand changes. To handle the increasing diversity and scalability of current applications, cloud environments offer resources with different characteristics and at different costs (based on duration of use).

Regardless of the complexity of the above 4 layers, it is necessary to optimize the performance of an application executing across the combined IoT-edge-cloud environment.

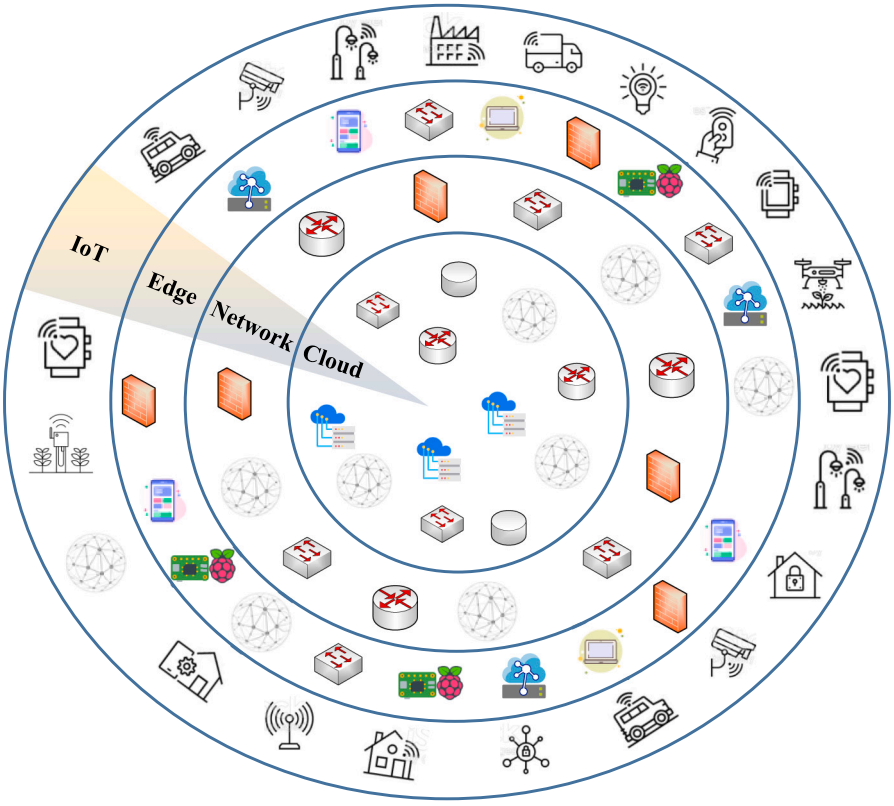


Fig. 1. 4-tier architecture — the outer layer is composed of IoT devices generating data and transmitting these to Edge devices (second layer). The Innermost layer is comprised of a Cloud datacenter, with a data network connecting these layers.

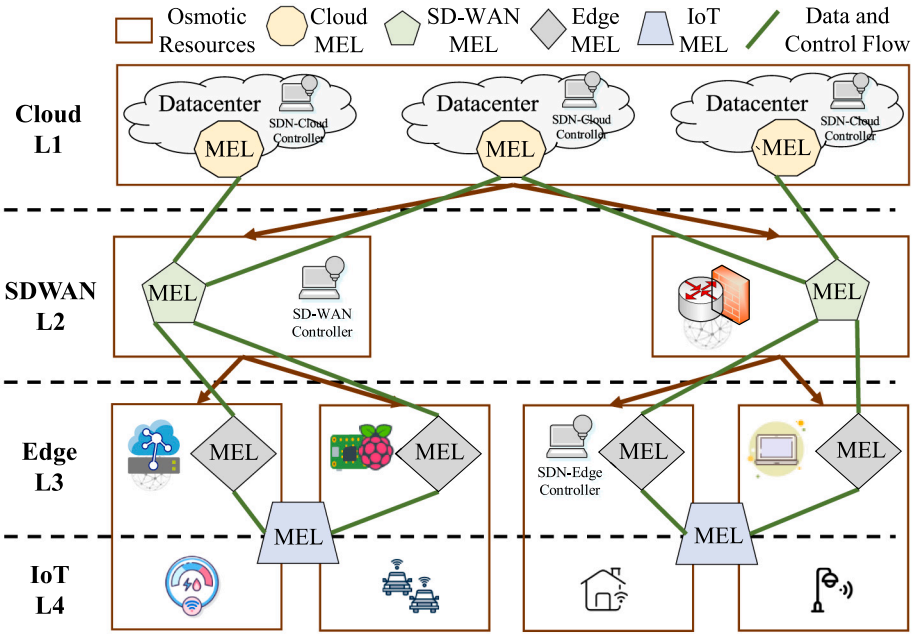


Fig. 2. Application MEL graph.

2.2. Application topology

Osmotic Computing focuses on strategies and mechanisms to extend IoT device capabilities by developing a computing model that makes use of all the 4 IoT infrastructure layers [18]. To handle the complexity and diversity of applications, it provides an abstraction referred to as "Microelements" (MELs) – which encapsulates services, resources and data. In particular, any IoT applications can be represented using a graph of MELs as shown in Fig. 2. Modeling an application as a graph of MELs involves:

- **Encapsulation of multiple components** In the context of an IoT application, sensed data needs to be processed across a number of *functions* or operations. The representation of each operation can take different forms, leading to an IoT application being specified

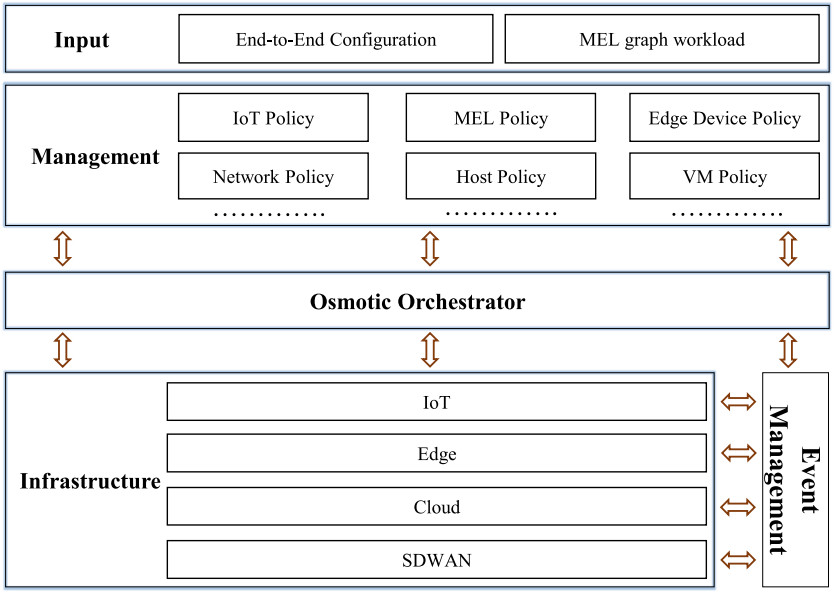


Fig. 3. Architecture of IoTSim-Osmosis simulator.

as a graph of MELs. Each MEL can contain micro data and be realized as a microservice which can be deployed on the IoT infrastructure. A MEL, as an entity, needs to abstract all of these capabilities.

- **Maintaining data and control flow** There is a strict dependency between various MELs within an application. The dependency can be in the form of data transfer or control flow. An example of MEL graph dependency is given in Fig. 2.
- **Performance optimization across heterogeneous IoT infrastructure** This involves understanding how the Cloud (L1) interacts and coordinates with the IoT (L4) and Edge (L3) layers, through an SDWAN (L2). Each MEL has specific QoS constraints limiting the locations at which a MEL can be deployed. For example a deep learning model cannot be deployed on IoT or edge device if it has specific QoS constraints. In addition to this, it is necessary to optimize the underlying IoT infrastructure layers while executing MELs.

3. Design of IoTSim-Osmosis

This section discusses the conceptual model of IoTSim-Osmosis, including its architecture and components.

3.1. IoTSim-Osmosis architecture

The architecture of IoTSim-Osmosis is presented in Fig. 3. It is divided into four main layers: *input*, *management*, *osmotic orchestrator* and *infrastructure*. IoTSim-Osmosis requires two *input* files — an end-to-end configuration file which includes a description of each infrastructure element. For example, it contains attributes of IoT device (e.g. device ID, bandwidth, battery capacity). When IoTSim-Osmosis finishes building the required infrastructure, it would require an IoT-MEL graph as workload to execute. The workload contains details of a *transaction*, represented as MELs and network operations. Each transaction can have different performance and can be used to evaluate the performance of a given osmotic application.

The *management* layer is modeled to facilitate the process of deploying tailor-made osmotic policies. It obtains several policies for different purposes e.g., network policy is designed to instruct SDN/SDWAN controllers with routing and traffic. As another example, virtual machine (VM) policy is used to select a host that can deploy requested VM. For each policy, IoTSim-Osmosis has a number of implemented algorithms that can be freely used.

The *infrastructure* layer is modeled to represent four infrastructure components: IoT, edge, cloud, and SDWAN. To provide a realistic representation of osmotic computing, each infrastructure component is modeled with many elements. For example, an IoT component is designed to obtain IoT devices with various attributes, such as device type, data rate, data type, and supporting network protocols. Finally, the *osmotic orchestrator* is designed to control all the events and operations happening in IoTSim-Osmosis. Using the event management system, it is able to manage the infrastructure and network while allowing a user to apply the management policies.

3.2. IoTSim-Osmosis system components

An overview of IoTSim-Osmosis's system components is illustrated in 4. IoTSim-Osmosis has an SDNSystem component, which mimics the general behavior of SDN. It is coupled with a routing table to store routing information and relation among nodes in its respective network. The child components (SDN-Edge controller, SDN-DC controller, and SDWAN controller) extend the SDNSystem to obtain general, shared functions along with adding their customized functions. Each controller obtains its unique route table, which is used to make proper routing decisions. An osmotic coordinator is used to interlink the controllers so that routing decisions are made in a global manner.

Each component of edge datacenter, cloud datacenter, and SDWAN is coupled with a topology component to describe the arrangement of the networks' nodes (e.g., edge devices, hosts, switches). Each component separately defines the way different nodes are interconnected with each other. The topology component is totally managed by a respective controller. Every controller must update its topology with network changes, such as an edge device is disconnected. Also, each controller uses its topology to help build routing tables.

Every edge datacenter has an associated proxy component — on an edge and IoT device. Similarly, the edge datacenter can have a number of connected IoT devices, generating data over a particular time interval. Each IoT device obtains a battery with an integrated consumption policy. Data from each IoT device is forwarded to a MEL component residing at an edge device. Every cloud datacenter maintains a number of hosts with associated MELs to carry out further processing when required.

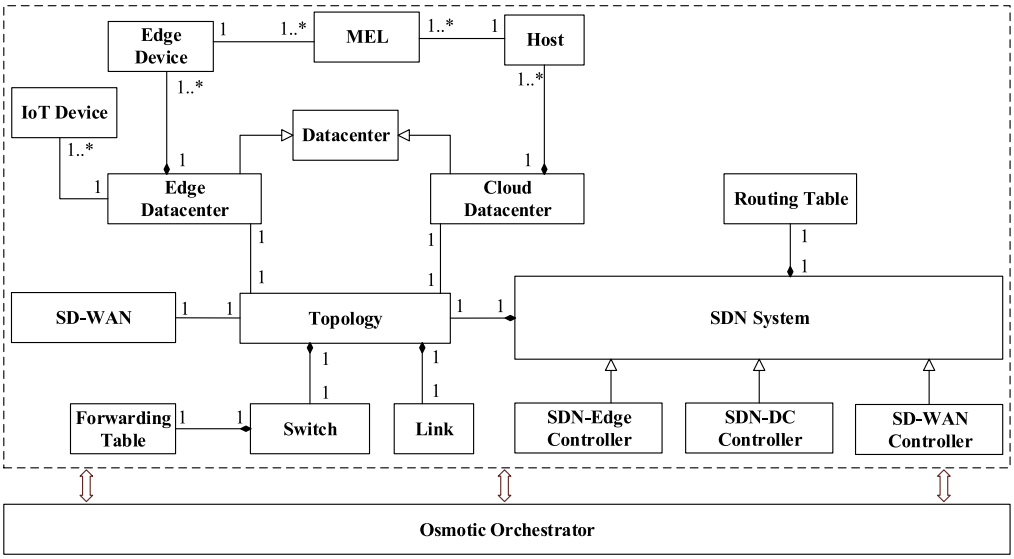


Fig. 4. IoTSim-Osmosis system components.

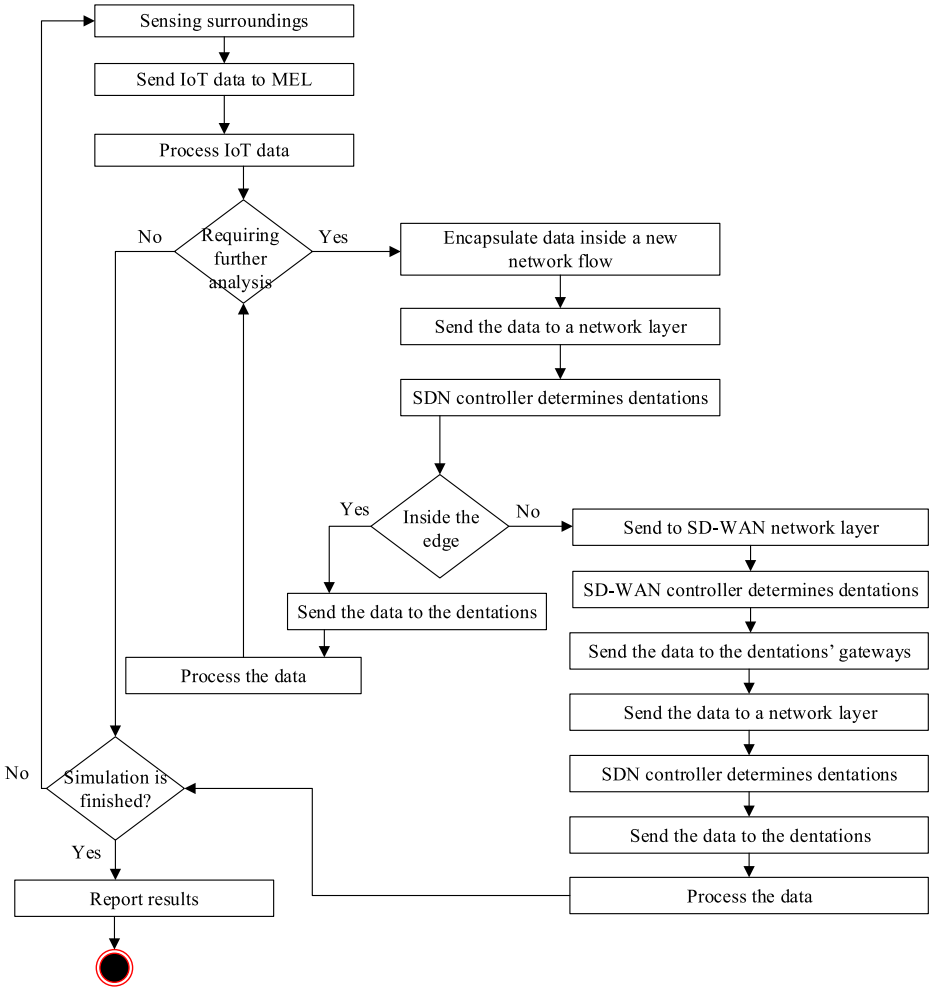


Fig. 5. IoTSim-Osmosis overview model.

3.3. IoTSim-Osmosis model

An overview of IoTSim-Osmosis's model is shown in Fig. 5. Every IoT device consistently senses its surrounding environment over a given time interval, sending its sensed data to a respective MEL residing in an edge datacenter. The MEL processes the received data, with the computational capability of MEL being specified in Million Instructions Per Seconds (MIPS). To support additional processing, data may be

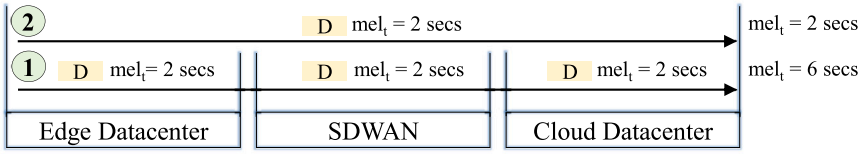


Fig. 6. Illustration of osmotic network transmission time.

exchanged with another MEL residing in the same edge datacenter or in another edge/cloud datacenter. The routing decision is handled by the SDN-edge controller. If the SDN-edge controller cannot determine the destined MEL, it will ask the source MEL to forward the data to the edge datacenter gateway. As an SDWAN controller is updated with network information of all associated datacenters by the osmotic orchestrator, it determines a path to the destined MEL. As a result, the SDWAN controller sends the data to the gateways of the destined MEL. As data arrives, the gateway requests its associated edge/cloud SDN controller to find a network route to the destined MEL (for data processing). The journey from IoT device to the last MEL is considered to be a *transaction* where every processing and transmission results are stored.

In general, data transmission in osmotic computing takes place multiple times based on a given application MEL graph. Any MEL graph always starts from an IoT layer where IoT devices observe and send their observed data to an associated edge MEL(s). To compute every IoT data transmission time iot_t, Eq. (1) is used where iot_{ds} is the IoT observed data size, iot_{bw} is the available bandwidth of an IoT device, and em_{bw} is the available bandwidth of an edge MEL. As the edge MEL might receive data from different IoT devices, it is important to take the minimum bandwidth of the two associated elements.

$$iot_t = \frac{iot_{ds}}{min(iot_{bw}, em_{bw})} \tag{1}$$

More data transmissions occur when edge MELs require further analysis by other edge/cloud MELs. It is important to compute a MEL data transmission time whenever a given MEL sends data to another MEL, whether in the same edge datacenter or in other edge/cloud datacenters. To properly compute every MEL data transmission time mel_t, an end-to-end network path must be first determined. Failure to do so would lead to incorrect estimation of mel_t. For example, Fig. 6 illustrates how the end-to-end network transmission time is computed incorrectly (step 1) and correctly (step 2). In step 1, it can be seen that each separate environment computes mel_t of the same data (D) and then mel_t are summed altogether. Each environment computes mel_t to 2 s, which result in mel_t = 6 s. Such estimation is incorrect because mel_t should be computed from a source MEL to a destined MEL rather than from one environment to another. The correct calculation is shown in step 2 where an end-to-end network estimation of mel_t is considered, which results in mel_t = 2 s.

In order to obtain an end-to-end osmotic network estimation for any given mel_t, an end-to-end path must first be established. Every edge, cloud, and SDWAN controller must communicate with one another via the event management component to establish the end-to-end path. Every controller has full control of its network where it selects the best path based on its routing algorithm (e.g., shortest path, maximum bandwidth). Following similar graph theory technique in Ref. [19], the path/routing table of every controller is dynamically determined. Once every controller determines its path, it sends the path information to the osmotic orchestrator. When the osmotic orchestrator has the end-to-end path information, it estimates the bandwidth of the end-to-end path $end_{bw}(x)$ for the xth MEL by using Eq. (2) where l denotes a link, L denotes a set of links, m is a decision variable set to 1 or 0 to determine if the link exists on the path or not respectively, and bw is the available bandwidth of l.

Next, the orchestrator requests the source MEL to send the data and in turn the orchestrator keeps estimating mel_t until the data is fully transmitted. To compute $mel_t(x)$ of the xth MEL, the orchestrator uses Eq. (3) where $mel_{ds}(x)$ is the data size of the xth source MEL,

$$end_{bw}(x) = min(l(m, bw)) \quad m = 1, \forall l \in L \tag{2}$$

$$mel_t(x) = \frac{mel_{ds}(x)}{end_{bw}(x)} \tag{3}$$

To compute the processing time of each MEL, Eq. (4) is used where $mel_e(t)$ is the processing time of the tth MEL, $mel_{mi}(t)$ is the Million Instruction (MI) size of the tth MEL, and $mel_{mi}(t)$ is the MIPS capacity of the tth MEL.

$$mel_e(t) = \frac{mel_{mi}(t)}{mel_{mips}(t)} \tag{4}$$

Eq. (5) is used to compete the total time of each transaction $\mathcal{T}$ where $\mathbb{X}$ is a set of MEL belongs to the transaction. The transaction is important to consider as it can determine the performance of each osmotic application.

$$\mathcal{T} = iot_t + \sum_{\forall x \in \mathbb{X}} mel_t(x) + mel_e(x) \tag{5}$$

IoTSim-Osmosis can be configured to stop generating IoT data at any given time. However, if the battery of all the IoT devices are drained, then IoTSim-Osmosis must stop and report the results. Therefore, to estimate the total running time $\mathcal{RT}(a)$ of the ath osmotic application, Eq. (6) is used where $tr_s(first)$ is the start time of the first transaction and $tr_f(last)$ is the finish time of the last transaction.

$$\mathcal{RT}(a) = tr_s(first) - tr_f(last) \tag{6}$$

As IoT devices might depend on batteries, IoTSim-Osmosis is modeled to track the battery consumption of each IoT device. Every time an IoT device senses new data, IoTSim-Osmosis would use Eq. (7) to update the battery consumption bc of the device where s_r is the draining rate for sensing the surrounding environment and t_r is the draining rate for sending the data. For computing the power consumption in edge, cloud, and SDWAN, IoTSim-Osmosis follows similar patterns as given in [20].

$$bc = s_r + t_r \tag{7}$$

4. Evaluation of IoTSim-Osmosis

A wide range of osmosis applications can be simulated and evaluated in IoTSim-Osmosis. This section illustrates the overall applicability of IoTSim-Osmosis in terms of simulating smart city applications based on the osmosis paradigm. The paradigm shift in traditional IoT environments to provide next-generation services and improves city infrastructures require a hybrid infrastructure that smartly interconnects IoT-oriented computing systems (SDN-enabled edge, SDN-enabled cloud, and SDWAN). IoTSim-Osmosis is developed to allow such hybrid infrastructure to be simulated where the dynamic management and performance metrics of IoT-oriented services across edge and cloud datacenters via SDWAN are easily achieved. The section provides strong evidence that IoTSim-Osmosis is an effective tool for assessing the effectiveness of tailor-made solutions for accelerating and enhancing the performance of heterogeneous osmosis applications.

Software availability: The IoTSim-Osmosis's software with the source code can be downloaded from https://github.com/kalwasel/IoTSim-

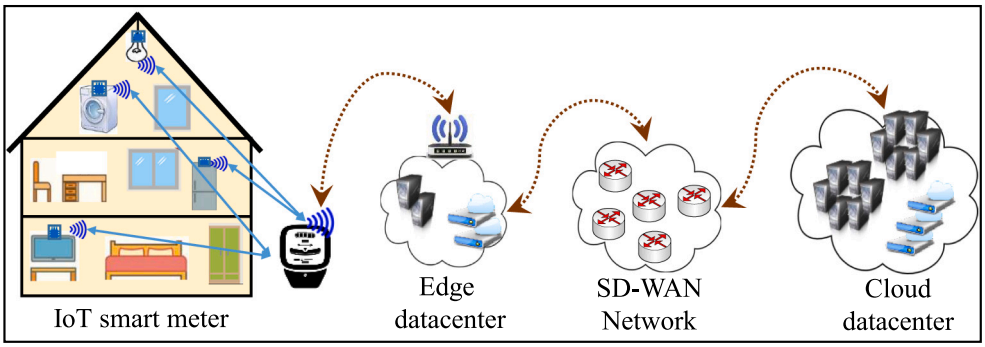


Fig. 7. Osmotic computing example (a smart home connected to a smart city electricity meter).

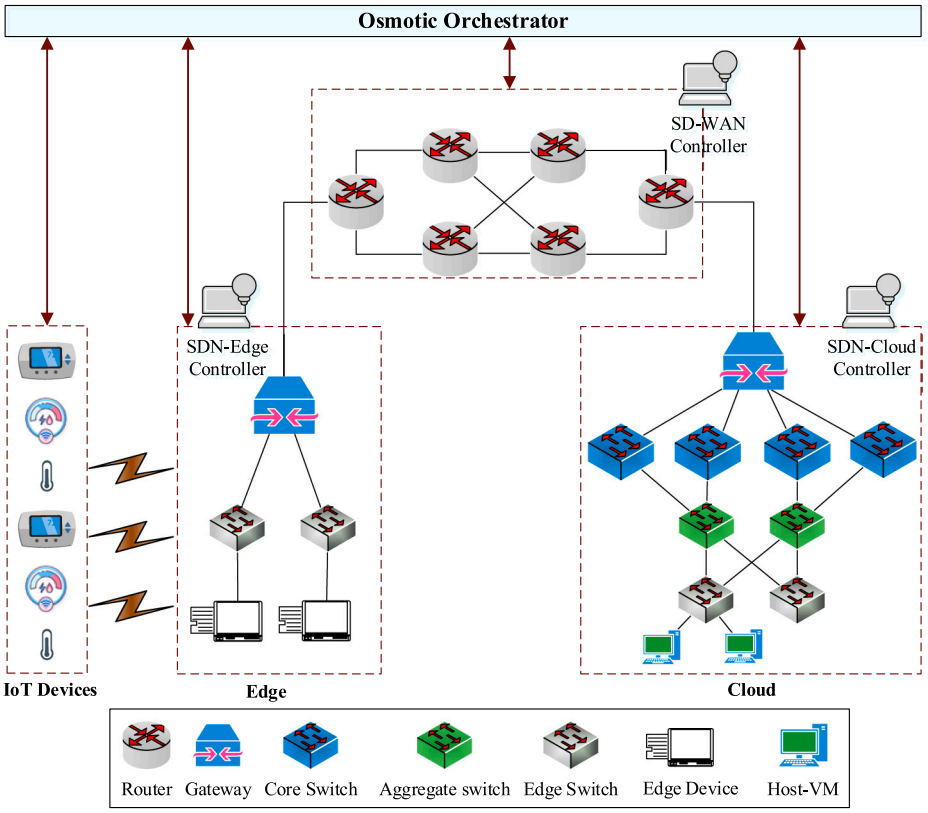


Fig. 8. Osmosis infrastructure for case 1.

Osmosis. A guideline for installation is given along with presenting a number of examples and tutorials to illustrate the use of the simulator. IoTSim-Osmosis uses features from a combination of existing simulation environments (IoTSim-SDWAN [19] and IoTSim-Edge [4]).

4.1. Smart city

The advances of IoT have contributed to the establishment of smart cities to improve citizens' quality of life. Developing a smart city requires the complex deployment of IoT ecosystems in numerous domains, such as in smart meters to save energy consumption, in roads to improve traffic management, and in self-driving cars to provide transportation for customers on demand. Each domain has various requirements (e.g., a certain level of communication delays, and artificial intelligence to enrich the decision-making process). Osmotic computing would play an essential role in enabling such requirements. It allows IoT applications to be defined in the form of MELs, which are deployed across several edge-cloud resources.

The example of electricity management and billing in smart city is used as an evaluation scenario. Several smart meter sensors installed in houses that collects data about the energy consumption. The sensor sends the data to a local gateway (edge device) installed nearby to perform basic analytic operation such as filtering and windowing. Since the smart meters can be of different types, we considered two scenarios, *(a) smart meter sensors with varying data rate (dynamic data flow)* and *(b) varying the number of smart meters (dynamic number of IoT devices)*. Finally the data is sent to cloud for further evaluation and storage. Fig. 7 illustrates an overview of a smart home connected to a smart city meter for electricity management and billing.

4.1.1. IoTSim-Osmosis policies

IoT-based osmotic applications and infrastructures require a number of policies in every layer of osmotic computing. IoTSim-Osmosis is modeled to support the implementation of new policies in a seamless manner where researchers can easily extend the main policies and develop tailor-made solutions and algorithms. Each layer can have

Table 1
Computing configuration for the use-cases.

IoT device		Edge device		Host (Cloud)		VM (Cloud)	
Bandwidth	100 Mbps	CPUs	4	CPUs	4	CPUs	2
Battery capacity	100 mA	Bandwidth	100 Mbps	Bandwidth	1 Gbps	Bandwidth	100 Mbps
Battery sensing rate	0.001 mAH	RAM	4 GB	RAM	8 GB	RAM	2 GB
Battery sending rate	0.001 mAH	MIPS/CPU	250	MIPS/CPU	1250	MIPS/CPU	250
Network type	WiFi	Storage	200 GB	Storage	500 GB	Storage	200 GB

Table 2
Network configuration for the use-cases.

Edge network		SDWAN network		Cloud network	
Edge device to edge switch	100 Mbps	Edge gateway to SDWAN router	100 Mbps	Gateway to aggregate switches	100 Mbps
edge switch to gateway	100 Mbps	Between SDWAN routers	100 Mbps	Core switches to aggregate switches	100 Mbps
–	–	Cloud gateway to SDWAN router	100 Mbps	Aggregate switches to edge switches	100 Mbps
–	–	–	–	Edge switches to VMs	100 Mbps

Table 3
Application configuration for case 1.

Tests	Data time interval (seconds)	Stop IoT data generation (seconds)	IoT device name	IoT Device output data (Mb)	MEL name	EdgeLet size	MEL output data (Mb)	VM name	CloudLet size
Test 1	10	3600	Variable	90	Variable	250	70	Variable	200
Test 2	15	3600	Variable	90	Variable	250	70	Variable	200
Test 3	20	3600	Variable	90	Variable	250	70	Variable	200
Test 4	25	3600	Variable	90	Variable	250	70	Variable	200

Table 4
Device requirement for case 1.

Number of IoT devices	Number of edge devices	Number of MELs	Number of hosts	Number of VMs
10	2	2	2	2

Table 5
Space and time complexity configuration.

Test	Number of IoT devices	Edge		Cloud			
		Number of datacenters	Number of edge devices	Number of datacenters	Number of edge devices	Number of hosts	Number of VMs
1	20	2	20	2	20	20	20
2	40	4	40	4	40	40	40
3	60	6	60	6	60	60	60
4	80	8	80	8	80	80	80
5	100	10	100	10	100	100	100

different policies; for example, the task scheduling of MELs in the edge can have a time-shared policy while VMs in the cloud can have a space-share mechanism. To properly execute IoTSim-Osmosis and illustrate the given use cases, the following policies are used:

- The task scheduling of MELs and VMs is based on a time-shared policy.
- The allocation of MELs and VMs is set to the least used resources of edge devices and cloud servers.
- Network routing in the edge, cloud, and SD-WAN is based on shortest-path maximum-bandwidth [19].
- The network traffic policy of IoT applications is based on a fair-share mechanism where each application obtains an equal amount of network bandwidth.

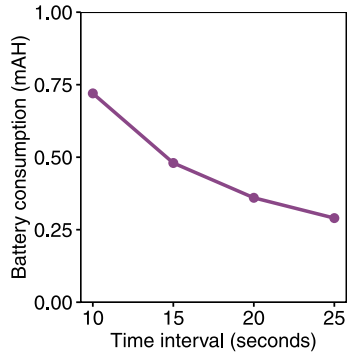
4.1.2. Case 1: dynamic data flow

This case is used to evaluate the outcome effectiveness of the simulator based on dynamic data intervals. The case is executed with four different data generation times. An overview of the simulated infrastructure setup is illustrated in Fig. 8. Table 1 shows the computing configuration of edge and cloud datacenters while Table 2 illustrates the network configuration in the edge, cloud, and SDWAN. Finally, Table 3 presents the attributes used to run each test. Table 4 shows the number of devices used in the case. The focus of this case is to show the effect of dynamic data generation intervals.

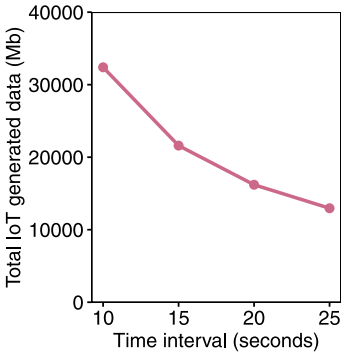
The simulation results are presented in Fig. 9. Fig. 9a illustrates the battery consumption of the IoT Devices. It can be observed that the lower the time interval for sending data, the higher the battery consumption. Fig. 9b shows the total size of the generated data by IoT devices, while Fig. 9c illustrates the total number of transactions. It can be seen that the total size of the generated data and transactions is inversely proportional to the size of the time interval. Fig. 9d shows the total time taken by each transaction. It can be observed that they consume similar time. This is because different transactions do not interfere with each other at any given resource (e.g., edge device, edge network). If the interval time is, for instance, set to one second, the time of each transaction would most likely vary. Fig. 9e shows the total energy consumption of edge, cloud, and SDWAN. The Figure reveals that the lower the time interval for IoT generating data, the higher the energy consumption. Fig. 9f shows the total time of transactions of every time interval. It is apparent that generating more data would lead to higher transaction times due requirement for more processing and transmission. Fig. 9g illustrates the total running/simulation time. The IoT devices are set to stop generating data at 3600 s. It can be seen that the finishing time is not similar because the last transaction of time intervals 5 and 6 requires more time to finish.

4.1.3. Case 2: dynamic number of IoT devices

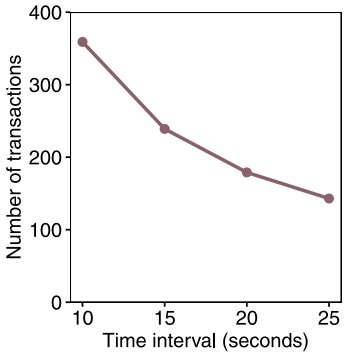
This case is used to evaluate the energy consumption of the osmotic environment by changing the number of associated IoT devices. The computing and network configurations are shown in Tables 1 and 2. This case has similar application configuration and device requirement as case 1 (see Tables 3 and 4). However, the number of IoT devices



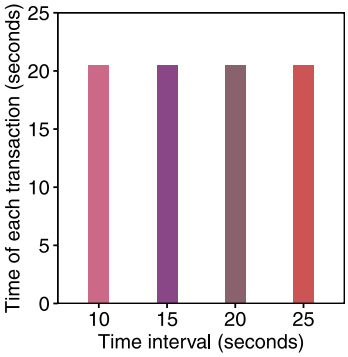
(a) Battery consumption



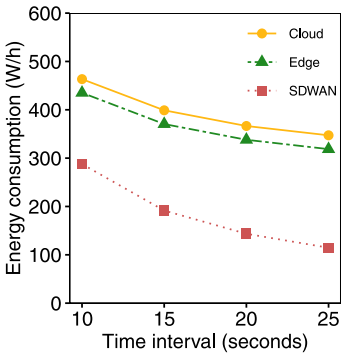
(b) Total IoT generated data



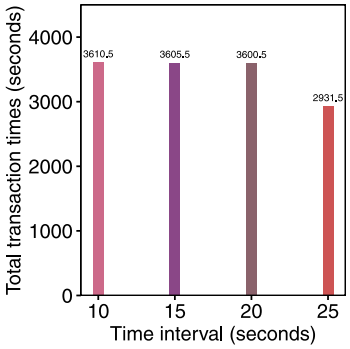
(c) Number of transactions



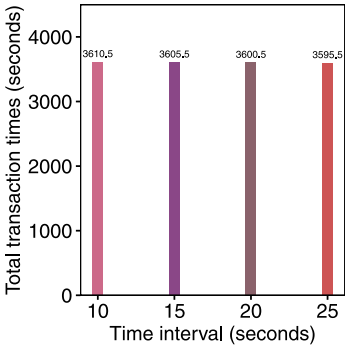
(d) Time of each IoT transaction



(e) Energy consumption of cloud, edge, and SDWAN infrastructures



(f) Total transaction times



(g) Total running time

Fig. 9. Simulation result for case 1.

varies from 2 to 10 to illustrate the impact of IoT devices on energy consumption of edge, cloud, and SDWAN. Also, the time interval for IoT devices to generate data is set to 10. Fig. 10 illustrates the energy consumption of edge, cloud, and SDWAN. It is apparent that the increase in the number of IoT devices requires more energy for the edge, cloud, and SDWAN. The battery consumption of IoT devices is neglected because data generation is static, which results in similar battery consumption for all the IoT devices.

4.2. Space and time complexity of IoTSim-Osmosis

The magnitude of memory and time consumption to simulate osmotic environments would vary from one case to another. A special scenario is presented to illustrate such consumption. The computing and network configurations are shown in Tables 1 and 2. The case has a similar application configuration as case 1 (see Table 3). However, the time interval for IoT devices to generate data is set to 20. The case special configuration is shown in Table 5. The case is executed five times, represented as tests. Fig. 11a illustrates the simulation time complexity. It can be seen that the simulation time increases as the number of requirement increases in each test. However, the simulation time is very reasonable for all tests, taking up to 68.70 s to complete the simulations. Fig. 11b shows the memory consumption. It can be observed that the memory consumption of the simulations slightly increases as the number of requirements increase in each test.

5. Related work

To simulate the complex environment of cloud, edge and underlying networks, various simulation and emulation frameworks have been introduced. This section summarizes the most relevant simulation and emulation frameworks and illustrates how these frameworks are not

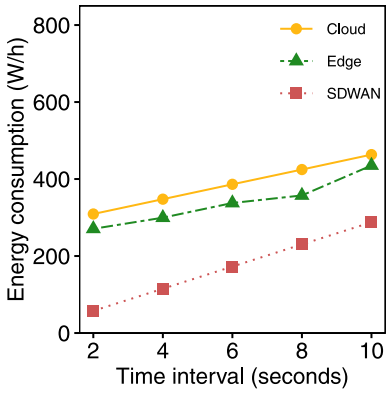


Fig. 10. Energy consumption of cloud, edge, and SDWAN infrastructures (use case 2).

able to satisfy the requirements for osmotic computing environments as compared to our proposed IoTSim-Osmosis simulator.

5.1. Cloud simulators

Multiple simulation frameworks have been proposed to model and simulate cloud computing infrastructures. The most popular one is CloudSim [21], which is a discrete-event simulation tool designed to enable the modeling and simulation of cloud-based systems and services. It supports the modeling of various cloud system components; for example, cloud datacenters, virtual machines (VMs) along with providing mechanisms to easily test and evaluate new strategies that improve the performance of cloud infrastructures. NetworkCloudSim [22] extends the functionality of CloudSim to leverage traditional network infrastructures within cloud datacenters. RC2Sim [23] is another cloud-based tool with the focus on evaluating cloud management techniques. It is a combination of simulation (e.g. calculating a time for creating a VM image) and emulation (e.g. sending real TCP/IP traffic) to enable the testing of large-scale cloud environments in a single machine.

iCanCloud [24] is also a cloud simulator offering several features for conducting large-scale cloud experiments. It can simulate computing and network resources efficiently. It is equipped with a global hypervisor to test different cloud brokering strategies. GreenCloud [25] is a cloud simulation toolkit built on top of an NS-2 simulator. It is capable of simulating computing and network cloud infrastructures along with offering numerous energy-aware models.

DCSim [26] is a cloud-based simulator that enables the modeling and simulating of cooling systems in addition to computing and network infrastructures. It is provides mechanisms to quantify the performance and energy consumption in terms of servers, network, and cooling systems. By using DCSim, energy-aware algorithms can be effectively evaluated. Multi-RECloudSim [27] is an extension of CloudSim focusing on modeling and simulating of multi-resource task executions. It provides rich features in terms of power modeling and multi-resource task scheduling. DISSECT-CF [28] is a customizable simulation framework which builds upon existing cloud computing concepts. It is mainly designed for energy consumption evaluation in relation to Infrastructure-as-a-Service (IaaS), which supports model task scheduling.

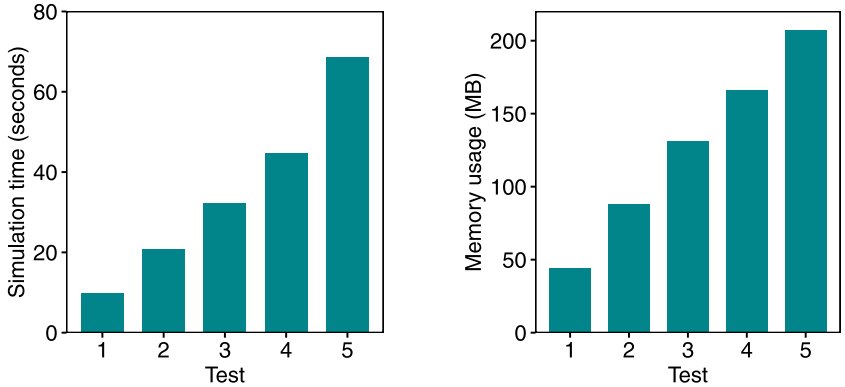
These simulators have the power to support modeling and simulation of cloud infrastructures, which include computing and traditional networking. However, they are limited to traditional clouds and do not simulate current technological paradigms (e.g., IoT, SDN, SD-WAN).

5.2. Network simulators and emulators

Several network-based simulation tools have been introduced for building and evaluating different types of network infrastructures in a simulated manner. Some examples of network infrastructures include wireless sensor networks (WSNs), local area networks (LANs), internet protocols (e.g. border gateway protocol). One of the most powerful network-based tools is NS-3 [29]. NS-3 is an open-source network simulator based on discrete-event mechanisms, which offers several types of network infrastructures, such as WSNs and LANs. It also provides several features, such as the ability to evaluate the designs and algorithms for the energy consumption and routing protocols of WSNs.

ConesC [30] is a verification WSN tool designed to easily deploy and test different types of WSN models in terms of design perspectives. It efficiently allows developers to check and evaluate the correctness of proposed WSN designs. COOJA [31] is a simulator that can be employed to model multiple deployment levels (e.g. operating systems, machine code instruction sets, and networks). Although COOJA is principally designed for use with the Contiki operating systems, it can also be used to support simulation of heterogeneous network nodes.

TOSSIM [32] is a toolkit that simulates the hardware components of sensor devices. It allows TinyOS applications to seamlessly run and interact with the underlying components of TOSSIM without the need for real sensor devices. TinyOS [44] is an operating system designed for wireless devices that are equipped with low-power batteries. By using TOSSIM, TinyOS applications can easily be evaluated and tested in terms of performance and energy consumption. OMNeT++ [33] is a generic network-based toolkit designed to simulate several network-specific domains/models (e.g., wireless ad-hoc network simulations, storage area network simulations). OMNeT++ has an effective graphical user interface (GUI) which accelerates and simplifies



(a) Time consumption (b) Memory consumption

Fig. 11. Simulation complexity.

Table 6
Comparison of various simulation frameworks with the proposed IoTSim-Osmosis.

Simulator	Features								
	Cloud processing	SDN support	SD-WAN support	Network comm.	Network protocols	Edge processing	Edge comm.	IoT devices	Application composition
CloudSim [21]	✓								
NetworkCloudSim [22]	✓			✓					
RC2Sim [23]	✓			✓					
iCanCloud [24]	✓			✓					
GreenCloud [25]	✓			✓					
DCSim [26]	✓			✓					
Multi-RECloudSim [27]	✓								
DISSECT-CF [28]	✓								
NS-3 [29]				✓	✓				
ConesC [30]				✓					
COOJA [31]				✓	✓				
TOSSIM [32]				✓	✓				
OMNeT++ [33]				✓	✓				
Castalia [34]				✓	✓				
GreenCastalia [35]				✓	✓				
Mininet [36]			✓	✓	✓				
CloudSimSDN [20]	✓	✓		✓					
BigDataSDNSim [37]	✓	✓		✓	✓				✓
IoTSim-SDWAN [19]	✓	✓	✓	✓	✓				
SDN-Sim [38]		✓		✓	✓				✓
SimIoT [6]	✓							✓	
Edge-Fog [39]						✓	✓	✓	
iFogSim [40]	✓					✓	✓	✓	
MyiFogSim [41]	✓			✓		✓	✓	✓	
EdgeCloudSim [42]						✓	✓	✓	
IoTSim-Edge [4]					✓	✓	✓	✓	✓
Diasuite [43]					✓			✓	✓
IoTsuite [5]								✓	✓
AWS IoT Device Simulator[1]	✓			✓				✓	✓
Microsoft IoT Simulator[2]	✓			✓				✓	✓
Proposed IoTSim-Osmosis	✓	✓	✓	✓	✓	✓	✓	✓	✓

the deployment of different network-based scenarios. Castalia [34] is as an extension of OMNet++ developed to simulate networks of low powered devices, such as body area networks. It can also be used to dynamically model and simulate large numbers of mobile nodes. GreenCastalia [35] extends the capability of Castilia to allow the modeling and simulation of harvesting-aware power management for embedded devices. The most important limitation of these simulators lies in the fact that they lack the support for SDN-aware mechanisms within and across edge-cloud environments.

5.3. SDN-aware network simulators and emulators

Mininet [36] is a lightweight SDN-centric emulation tool that enables virtualization mechanisms for large-scale SDN-aware networks in a single machine. It offers the advantage of quantifying SDN performance within different network structures and routing protocols. CloudSimSDN [20] extends the functionality of CloudSim to provide SDN architectures and models within cloud datacenters. Additionally, it consists of different network and management strategies for energy management. BigDataSDNSim [37] is built on top of cloudSimSDN and provides models to derive different performance and network metrics of big data applications in SDN-enabled cloud datacenters.

IoTSim-SDWAN [19] is a new simulation tool that provides a model of distributed SDN-enabled cloud datacenters communicating via SD-WAN network infrastructures. It facilitates the process of evaluating new designs and algorithms in the context of SD-WAN/SDN aware datacenters. SDN-Sim is a new simulator and emulation toolkit that integrates multiple frameworks (e.g., OpenDaylight SDN controller, Mininet, and GNS-3). It supports different SDN-based simulation and emulation models to evaluate different SDN performance perspectives. The focus of SDN-Sim [38] is to facilitate the deployment and testing of several SDN-based policies, such as channel modeling, traffic shaping, and QoS demands.

5.4. IoT, edge, and fog simulators

In recent years, several simulators have been proposed to simulate IoT and edge environments. SimIoT [6] is another simulator which operates by modeling the transmission of data between IoT devices and cloud datacenters. Whilst the simulations associated with this tool do not include edge devices, it permits the dynamic testing of multi-user submissions in IoT contexts. Another simulator, Edge-Fog [39] supports various energy and network models in addition to assisting with task scheduling. iFogSim [40] can be employed for modeling IoT and Fog environments where all the computing nodes are represented as fog nodes. Moreover, it measures the influence of resource managements in relation to network congestion, cost, latency, and energy use. MyiFogSim [41] extends iFogSim and simulate network configurations, failures, and provisioning of mobile customers according to given virtual machine migration policies.

EdgeCloudSim [42] and IoTSim-Edge [4] extends the capability of CloudSim to incorporate different features of IoT and edge computing environments. While EdgeCloudSim explores the modeling of network links, mobility, and edge servers, it lacks IoT application composition and network complexities. IoTSim-Edge handles the application complexity along with heterogeneous communication mechanisms and mobility. However, both these simulators does not support the cloud and SD-WAN layers, which are essential components of IoT infrastructures.

A number of IoT-based simulators are also proposed for the deployment and testing of IoT applications. Diasuite [43] and IoTsuite [5] are the most common frameworks for managing the whole lifecycle of IoT applications. Both rely on a Siafu [45] simulator for evaluating proposed solutions for IoT applications. However, these frameworks have a very limited support for handling the complexity and deployment of IoT applications and infrastructures. Few industry-oriented

simulators are also available (e.g., AWS IoT Device Simulator[1] and Microsoft IoT Simulator[2]). The Amazon Web Services (AWS) simulator can be executed only on AWS infrastructures (where user have to pay) while the Microsoft simulator can be used only on a Windows 10 environment. There are limits to how far they support networking and SDN-aware environments. Also, defining and evaluating various end-to-end IoT policies and algorithms are complex in these industry simulators.

In summary, there are numerous frameworks available for simulating cloud, edge and/or SDN-based network components. However, none of the existing frameworks simulate the composition of all these components along with abstraction of complex IoT applications. Our proposed simulator, IoTSim-Osmosis covers all these components in a holistic manner and provides researchers the necessary support to evaluate end-to-end IoT application performance using the concept of osmotic computing. The advantage of IoTSim-Osmosis as compared with the existing simulation frameworks is clearly visible in Table 6.

6. Conclusions and future work

Osmotic computing provides a simplified model for the deployment of IoT applications in the integrated edge-cloud environment. This paper propose a simulation framework, IoTSim-Osmosis for analyzing and validating the osmotic computing environment in a simple manner. In particular, IoTSim-Osmosis handles the heterogeneity of integrated edge-cloud environments along with the complexity of IoT applications. The efficacy of IoTSim-Osmosis is validated using a case study for an electricity management and billing application within a smart city. Results show the various capabilities of IoTSim-Osmosis in terms of IoT battery, execution time, and energy consumption. Our results also demonstrate the scalability of IoTSim-Osmosis in terms of time and memory consumption.

Our experimental results and related work analysis demonstrate the useful and unique simulation capabilities of IoTSim-Osmosis. Future work will focus on enhancing IoTSim-Osmosis capabilities in a number of directions. The modeling of its IoT protocols (e.g., XMPP), although not the focus of this paper, is currently very basic. Therefore, further research will be conducted to investigate and model IoT protocols according to their characteristics and functionalities. Moreover, the modeling of wireless communication (e.g., WiFi) is currently limited to bandwidth speed, and the current implementation of IoTSim-Osmosis assumes IoT devices to be in fixed locations. We will extend the wireless communication layer of IoTSim-Osmosis to include different signal factors, such as distance and IoT device mobility.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work in this paper is supported by Saudi Electronic University (SEU) through the Saudi Arabian Culture Bureau (SACB) in the United Kingdom. This research is also supported by three UK projects, SUPER: EP/T021985/1, PACE: EP/R033293/1 and Osmotic MindSphere: P35792/BH192113.

[1] https://aws.amazon.com/solutions/implementations/iot-device-simulator/ .

[2] https://www.microsoft.com/en-us/p/iot-simulator/ .

References

[1] D.N. Jha, P. Michalak, Z. Wen, P. Watson, R. Ranjan, Multi-objective deployment of data analysis operations in heterogeneous iot infrastructure, IEEE Trans. Ind. Inf. (2019).

[2] M. Villari, M. Fazio, S. Dustdar, O. Rana, R. Ranjan, Osmotic computing: A new paradigm for edge/cloud integration, IEEE Cloud Comput. 3 (6) (2016) 76–83.

[3] M. Autili, P. Inverardi, M. Tivoli, Choreos: large scale choreographies for the future internet, in: 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), IEEE, 2014, pp. 391–394.

[4] D.N. Jha, K. Alwasel, A. Alshoshan, X. Huang, R.K. Naha, S.K. Battula, S. Garg, D. Puthal, P. James, A. Zomaya, et al., Iotsim-edge: A simulation framework for modeling the behavior of internet of things and edge computing environments, Softw. - Pract. Exp. 50 (6) (2020) 844–867.

[5] S. Chauhan, P. Patel, A. Sureka, F.C. Delicato, S. Chaudhary, IoTSuite: a framework to design, implement, and deploy IoT applications: demonstration abstract, in: Proceedings of the 15th International Conference on Information Processing in Sensor Networks, 2016, pp. 1–2.

[6] S. Sotiriadis, N. Bessis, E. Asimakopoulou, N. Mustafee, Towards simulating the internet of things, in: 2014 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE, 2014, pp. 444–448.

[7] G. Kecskemeti, G. Casale, D.N. Jha, J. Lyon, R. Ranjan, Modelling and simulation challenges in internet of things, IEEE Cloud Comput. 4 (1) (2017) 62–69.

[8] O. Salman, I. Elhajj, A. Chehab, A. Kayssi, Iot survey: An sdn and fog computing perspective, Comput. Netw. 143 (2018) 221–246.

[9] A.A. Barakabitze, A. Ahmad, R. Mijumbi, A. Hines, 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges, Comput. Netw. 167 (2020) 106984.

[10] A. Brogi, S. Forti, Qos-aware deployment of iot applications through the fog, IEEE Internet Things J. 4 (5) (2017) 1185–1192.

[11] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): A vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645–1660.

[12] M. Tao, J. Zuo, Z. Liu, A. Castiglione, F. Palmieri, Multi-layer cloud architectural model and ontology-based security service framework for iot-based smart homes, Future Gener. Comput. Syst. 78 (2018) 1040–1051.

[13] I. Lee, K. Lee, The internet of things (iot): Applications, investments, and challenges for enterprises, Bus. Horiz. 58 (4) (2015) 431–440.

[14] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, M. Nemirovsky, Key ingredients in an iot recipe: Fog computing, cloud computing, and more fog computing, in: Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 IEEE 19th International Workshop on, IEEE, 2014, pp. 325–329.

[15] W. Shi, S. Dustdar, The promise of edge computing, Computer 49 (5) (2016) 78–81.

[16] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: Experience with a globally-deployed software defined wan, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 3–14.

[17] W. Yu, F. Liang, X. He, W.G. Hatcher, C. Lu, J. Lin, X. Yang, A survey on the edge computing for the internet of things, IEEE Access 6 (2017) 6900–6919.

[18] M. Villari, M. Fazio, S. Dustdar, O. Rana, D.N. Jha, R. Ranjan, Osmosis: The osmotic computing platform for microelements in the cloud, edge, and internet of things, Computer 52 (8) (2019) 14–26.

[19] K. Alwasel, D.N. Jha, E. Hernandez, D. Puthal, M. Barika, B. Varghese, S.K. Garg, P. James, A. Zomaya, G. Morgan, et al., Iotsim-sdwan: A simulation framework for interconnecting distributed datacenters over software-defined wide area network (sd-wan), J. Parallel Distrib. Comput. (2020).

[20] J. Son, A.V. Dastjerdi, R.N. Calheiros, X. Ji, Y. Yoon, R. 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.

[21] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A. De Rose, R. 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.

[22] S.K. Garg, R. Buyya, Networkcloudsim: Modelling parallel applications in cloud simulations, in: 2011 Fourth IEEE International Conference on Utility and Cloud Computing, IEEE, 2011, pp. 105–113.

[23] D. Citron, A. Zlotnick, Testing large-scale cloud management, IBM J. Res. Dev. 55 (6) (2011) 1–10.

[24] A. Núñez, J.L. Vázquez-Poletti, A.C. Caminero, G.G. Castañé, J. Carretero, I.M. Llorente, Icancloud: A flexible and scalable cloud infrastructure simulator, J. Grid Comput. 10 (1) (2012) 185–209.

[25] D. Kliazovich, P. Bouvry, S.U. Khan, Greencloud: a packet-level simulator of energy-aware cloud computing data centers, J. Supercomput. 62 (3) (2012) 1263–1283.

[26] M. Tighe, G. Keller, M. Bauer, H. Lutfiyya, Dcsim: A data centre simulation tool for evaluating dynamic virtualized resource management, in: 2012 8th International Conference on Network and Service Management (Cnsm) and 2012 Workshop on Systems Virtualiztion Management (Svm), IEEE, 2012, pp. 385–392.

[27] W. Lin, S. Xu, L. He, J. Li, Multi-resource scheduling and power simulation for cloud computing, Inform. Sci. 397 (2017) 168–186.

[28] G. Kecskemeti, Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds, Simul. Model. Pract. Theory 58 (2015) 188–218.

[29] T.R. Henderson, M. Lacage, G.F. Riley, C. Dowell, J. Kopena, Network simulations with the ns-3 simulator, SIGCOMM Demonstr. 14 (14) (2008) 527.

[30] M. Afanasov, L. Mottola, C. Ghezzi, Software adaptation in wireless sensor networks, ACM Trans. Auton. Adapt. Syst. (TAAS) 12 (4) (2018) 1–29.

[31] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, T. Voigt, Cross-level sensor network simulation with cooja, in: Proceedings. 2006 31st IEEE Conference on Local Computer Networks, IEEE, 2006, pp. 641–648.

[32] P. Levis, N. Lee, M. Welsh, D. Culler, Tossim: Accurate and scalable simulation of entire tinyos applications, in: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, 2003, pp. 126–137.

[33] A. Varga, R. Hornig, An overview of the omnet++ simulation environment, in: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, ICST (Institute for Computer Sciences, Social-Informatics and . . . , 2008, p. 60.

[34] A. Boulis, Castalia: revealing pitfalls in designing distributed algorithms in wsn, in: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, 2007, pp. 407–408.

[35] D. Benedetti, C. Petrioli, D. Spenza, GreenCastalia: An energy-harvesting-enabled framework for the Castalia simulator, in: Proceedings of the 1st International Workshop on Energy Neutral Sensing Systems, 2013, pp. 1–6.

[36] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, 2010, pp. 1–6.

[37] K. Alwasel, R.N. Calheiros, S. Garg, R. Buyya, M. Pathan, D. Georgakopoulos, R. Ranjan, Bigdatasdnsim: A simulator for analyzing big data applications in software-defined cloud data centers, Softw. - Pract. Exp. (2020).

[38] S. Ghosh, S. Busari, T. Dagiuklas, M. Iqbal, R. Mumtaz, J. Gonzalez, S. Stavrou, L. Kanaris, Sdn-sim: Integrating a system-level simulator with a software defined network, IEEE Commun. Stand. Mag. 4 (1) (2020) 18–25.

[39] N. Mohan, J. Kangasharju, Edge-fog cloud: A distributed cloud for internet of things computations, in: 2016 Cloudification of the Internet of Things (CIoT), IEEE, 2016, pp. 1–6.

[40] H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, R. Buyya, Ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments, Softw. - Pract. Exp. 47 (9) (2017) 1275–1296.

[41] M.M. Lopes, W.A. Higashino, M.A. Capretz, L.F. Bittencourt, Myifogsim: A simulator for virtual machine migration in fog computing, in: Companion Proceedings of the10th International Conference on Utility and Cloud Computing, 2017, pp. 47–52.

[42] C. Sonmez, A. Ozgovde, C. Ersoy, Edgecloudsim: An environment for performance evaluation of edge computing systems, Trans. Emerg. Telecommun. Technol. 29 (11) (2018) e3493.

[43] B. Bertran, J. Bruneau, D. Cassou, N. Loriant, E. Balland, C. Consel, Diasuite: A tool suite to develop sense/compute/control applications, Sci. Comput. Program. 79 (2014) 39–51.

[44] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al., Tinyos: An operating system for sensor networks, in: Ambient Intelligence, Springer, 2005, pp. 115–148.

[45] M. Martin, P. Nurmi, A generic large scale simulator for ubiquitous computing, in: 2006 Third Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services, IEEE, 2006, pp. 1–3.