Newcastle University

School of Computing Science

Modeling and Simulation of Data-Driven Applications in SDN-aware Environments

Khaled Alwasel

Submitted for the degree of Doctor of Philosophy in the School of Computing, Newcastle University

December 2020

Abstract

The rising popularity of Software-Defined Networking (SDN) is increasing as it promises to offer a window of opportunity and new features in terms of network performance, configuration, and management. As such, SDN is exploited by several emerging applications and environments, such as cloud computing, edge computing, IoT, and data-driven applications. Although SDN has demonstrated significant improvements in industry, still little research has explored the embracing of SDN in the area of cross-layer optimization in different SDN-aware environments.

Each application and computing environment require different functionalities and Quality of Service (QoS) requirements. For example, a typical MapReduce application would require data transmission at three different times while the data transmission of stream-based applications would be unknown due to uncertainty about the number of required tasks and dependencies among stream tasks. As such, the deployment of SDN with different applications are not identical, which require different deployment strategies and algorithms to meet different QoS requirements (e.g., high bandwidth, deadline). Further, each application and environment has unique architectures, which impose different form of complexity in terms of computing, storage, and network. Due to such complexities, finding optimal solutions for SDN-aware applications and environments become very challenging.

Therefore, this thesis presents multilateral research towards optimization, modeling, and simulation of cross-layer optimization of SDN-aware applications and environments. Several tools and algorithms have been proposed, implemented, and evaluated, considering various environments and applications[1–4]. The main contributions of this thesis are as follows:

- Proposing and modeling a new holistic framework that simulates MapReduce applications, big data management systems (BDMS), and SDN-aware networks in cloud-based environments. Theoretical and mathematical models of MapReduce in SDN-aware cloud datacenters are also proposed.

- Proposing and modeling a novel framework that simulates SD-WAN and distributed SDN-aware datacenters. The framework models several network components, including TCP and UDP protocols and network delays. A new SD-WAN routing technique is proposed based on graph theory, which dynamically computes the best route for every network flow. A new coordination technique for SDN and SD-WAN controllers is also proposed to coordinate network functionalities in a global sense.

- Proposing and modeling a new SD-WAN based Workflow Broker (SDWAN-WB) to deploy the workflows of data-driven applications across multiple SD-WAN-enabled datacenters. An adaptive Genetic Algorithm (GA) is also proposed for selecting near-optimal solutions based on green energy usage, the topology and deadline of data-driven workflows, and overall energy consumption and cost.

- Proposing a unified osmotic computing framework that models and simulates complex IoT applications running over heterogeneous edge-cloud environments. Theoretical and mathematical models of osmotic computing are also proposed, in addition to several system management policies.

Publications

1. U. Demirbaga, Z. Wen, A. Noor, K. Mitra, **K. Alwasel**, S. Garg, A. Zomaya and R. Ranjan, "AutoDiagn: An Automated Real-time Diagnosis Framework for Big Data Systems," *IEEE Transactions on Computers*, April, 2021.

2. Z. Wen, S. Garg, G. Aujla, **K. Alwasel**, D. Puthal, S. Dustdar, A. Zomaya, and R. Rajan, "Running Industrial Workflow Applications in a Software-defined Multi-Cloud Environment using Green Energy Aware Scheduling Algorithm," *IEEE Transactions on Industrial Informatics*, December, 2020.

3. **K. Alwasel**, D. Jha, F. Habeeb, O. Rana, Thar Baker, S. Dustdar, M. Villari, P. James, E. Solaiman, and R. Ranjan, "IoTSim-Osmosis: A Framework for Modeling and Simulating IoT Applications over an Edge-Cloud Continuum," *Journal of Systems Architecture*, November, 2020.

4. A. Alqahtani, **K. Alwasel**, A. Noor, K. Mitra, E. Solaiman, and R. Ranjan. "The Integration of Scheduling, Monitoring, and SLA in Cyber Physical Systems." *In Handbook of Integration of Cloud Computing, Cyber Physical Systems and Internet of Things*, pp. 237-254. Springer, Cham, November 2020.

5. **K. Alwasel**, R. N. Calheiros, S. K. Garg, R. Buyya, and R. Ranjan, "BigDataSDNSim: A Simulator for Analyzing Big Data Applications in Software-Defined Cloud Datacenters," *Journal of Software: Practice and Experience*, October, 2020.

6. **K. Alwasel**, D. Jha, E. Hernandez, D. Putha, M. Barika, B. Varghese, S. Garg, P. James, A. Zomaya, G. Morgan, and R. Ranjan, "IoTSim-SDWAN: A Simulation Framework for Interconnecting Distributed Datacenters over Software-Defined Wide Area Network (SD-WAN)," *Journal of Parallel and Distributed Computing*, May, 2020.

7. D. N. Jha, **K. Alwasel**, A. Alshoshan, X. Huang, R. Naha, S. Battula, S. Garg, D. Puthal, P. James, A. Zomaya, S. Dustdar, and R. Ranjan, "IoTSim-Edge: A simulation framework for modeling the behavior of Internet of Things and edge computing environments," *Journal of Software: Practice and Experience*, January, 2020.

8. **K. Alwasel**, Y. Li, P. Jayaraman, S. K. Garg, R. N. Calheiros, and R. Ranjan, "Programming SDN-Native Big Data Applications: Research Gap Analysis," *IEEE Cloud Computing*, October 2017.

9. **K. Alwasel**, A. Noor, Y. Li, E. Solaiman, S. K. Garg, P. P. Jayaraman, and R. Ranjan, "Cloud Resource Scheduling, Monitoring, and Configuration Management in the Software Defined Networking Era," *IEEE Technical Committee on Cybernetics for Cyber-Physical Systems*, Volume 2, Issue 1, February, 2017.

Declaration

I declare that this thesis is my own work unless otherwise stated. No part of this thesis has previously been submitted for a degree or any other qualification at Newcastle University or any other institution.

Khaled Alwasel

December 2020

Acknowledgements

First, I would like to express my sincere gratitude and appreciation to my main supervisor **Prof. Rajiv Ranjan** for his extraordinary support and guidance since the first day of my PhD journey. His supportive personality, effective communication, and continuous constructive feedback have played significant roles in my PhD completion. Furthermore, I would like to extend my gratitude as well to Dr. Ellis Solaiman for his rich insights and feedback, which were much needed to facilitate things forward.

In addition to my supervisors, I would like to extend my thanks to Dr. Blesson Varghese, Prof. Rajkumar Buyya, Dr. Rodrigo N Calheiros, and Dr. Saurabh Garg for their collaborations and mentorship at different stages of my PhD research and publications.

I would like to thank my lab-mates: Ayman Noor, Devki Nandan Jha, Nipun Balan, Top Phengsuwan, Umit Demirbaga, Yinhao Li, and Zhenyu Wen for their encouragement, support, and friendship during my PhD candidature at Newcastle University.

Last but not least, I would like to thank the government of Saudi Arabia represented by Saudi Electronic University (SEU) and the Royal Embassy of Saudi Arabia Cultural Bureau for funding my PhD candidature. Without their support, this PhD would not have been possible.

The success of my PhD journey would not have been possible without the unconditional support and encouragement of my parents, family, and friends. To my dad who passed away in the last year of my PhD due to coronavirus, I thank you for your unwavering support and I appreciate you more than words can ever say.

Contents

List of Figures

List of Tables

1

Introduction

Contents

Introduction

Network systems are the core components that interconnect devices to each other for the purpose of sharing data. Constant development and improvement of network systems are essential to accelerate data delivery and satisfy modern applications' networking requirements. Recently, Software-Defined Networking (SDN) has emerged as a game-changer of traditional networking systems [7, 8]. It is a new paradigm aiming to address the real-time programmability shortcomings of traditional network systems. It has evolved as a new network model to dynamically program, control, change, and manage networking components in real-time [9]. As such, SDN has been deployed in various technological domains (e.g., cloud datacenters, edge datacenters) to facilitate network management and satisfy the quality of services (QoS) requirements [10, 11]. Figure 1.1 illustrates an overview of SDN usage and deployment in several computing environments (e.g., IoT-edge, cloud, and SD-WAN).

The rising popularity of SDN is increasing as it promises to offer a window of opportunity and great values to several emerging applications and environments. Data-driven applications are among the most beneficiaries to take advantage of SDN capabilities within cloud datacenters [3, 12, 13]. Such applications require various operations (e.g., capturing, transmission, storing, processing) and several big data programming models to effectively analyze large-scale data-sets at an astonishing speed. Big data has become the most prominent mechanism to delve into data-sets that are generated by data-driven applications [14, 15]. It harnesses the power of clustering commodity hardware to carry out data analysis in a parallel and simultaneous manner. The coordination and integration of SDN and big data in cloud datacenters would offer unprecedented advantages for data-driven applications, such as improving their network topology configuration and data transmission at run-time.

SDN paradigm has been also exploited by Wide Area Network (WAN) to obtain SDN functionalities [16, 17]. The traditional WAN is a core network layer that provides the fundamental building blocks for secure and salable shared resource access across geographically dispersed distributed systems [18]. The merging of WAN and SDN is referred to as a Software-defined Wide Area Network (SD-WAN). The SD-WAN

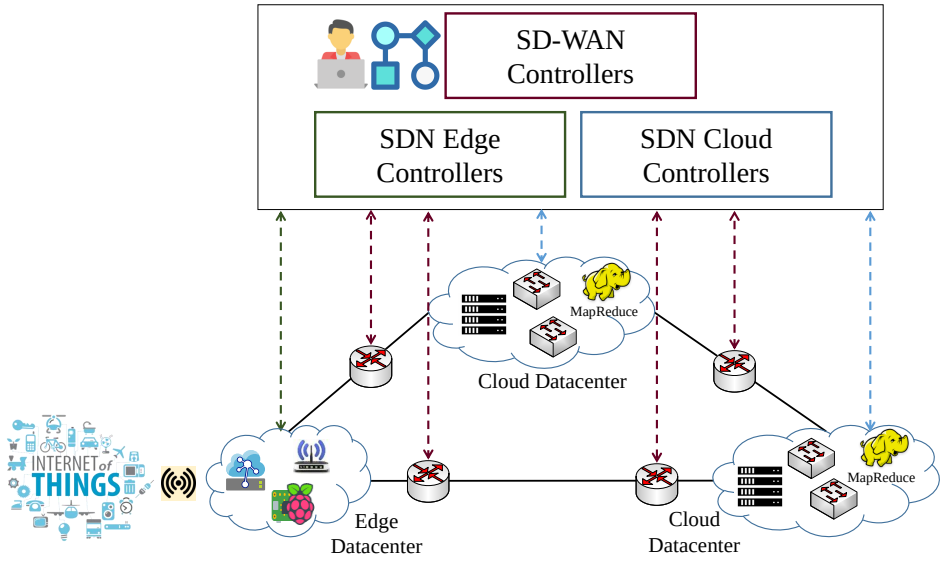


Figure 1.1: An overview of SDN usage and deployment in several environments

paradigm enables distributed cloud datacenters to gain a high level of resource management and efficiency, especially for data-driven applications, such as Internet of Things (IoT) applications and distributed gaming applications [19].

The leveraging of SDN's power is also exploited by edge computing and cellular networks (e.g., 5G) to facilitate the management of edge-cellular networks and accelerate the processing of IoT applications [20, 21]. An edge infrastructure significantly cuts the time it takes to process and analyze IoT generated data for specific applications as it offers processing mechanisms near IoT devices and sensors [22]. It also eliminates the amount of SD-WAN network traffic between edge and cloud. By adopting SDN, edge datacenters can seamlessly determine network utilization behavior, which results in an informed network decisions. SDN would also add remarkable benefits for IoT-based data-driven applications running across edge-cloud environments, such as finding optimal network routes at run-time [23, 24].

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 IoT ecosystems and workflow analytics – has led to a new generation of heterogeneous and complex environments. Such transition can be defined as *osmotic computing paradigm* [25]. The osmotic computing paradigm provides a simplified model for the deployment of IoT applications and workflows in the integrated edge-cloud en-

vironment. It focuses on the design and implementation of a unified computing model that leverages the capabilities of various distributed systems, which include IoT, edge computing, cloud computing, SDN, and SD-WAN. The coordination and integration of osmotic computing components empowered by SDN capabilities are essential to optimize the performance of data-driven IoT applications.

As shown in Figure 1.1, this thesis focuses on SDN usage and deployment in several emerging applications and computing environments. In the figure, it can be seen that there are multiple SDN-aware controllers implemented in IoT-edge, cloud, and SD-WAN environments. Starting from the IoT-edge, the SDN controller is responsible for managing edge internal network along with incoming IoT traffic. When IoT-edge data require further processing, it would forward data to cloud datacenters via SD-WAN layer. SD-WAN controllers are used to manage network between distributed edge/cloud datacenters. SDN cloud controllers are used to manage cloud internal networks, such as enforcing real-time instructions and rules on network devices. Further, IoT-edge data might require complex big data models to accelerate processing time; as such, MapReduce big data applications are essential to be implemented in cloud datacenters to deliver such processing requirements. To this end, this thesis studies how to effectively model, optimize, integrate, and simulate SDN within different data-driven applications and infrastructures, which includes IoT-edge, cloud, SD-WAN, and MapReduce big data applications.

1.1 Research Motivation

Although the SDN paradigm provides well-defined communication standards and application programming interfaces (APIs), the availability of software frameworks and algorithmic techniques to dynamically manage and optimize the performance of SDN-aware applications and SDN-aware computing environments is still nascent. SDN imposes several challenges that require a variety of solutions based on the requirements of different applications and infrastructures (e.g., performance optimization, automatic configuration, energy-saving, bandwidth allocation, 5G slicing, security assurance) [26–29]. Every challenge has a different form of complexity based on given

requirements (e.g., high bandwidth, deadline), which requires a unique solution.

This thesis considers an overall question of *how to optimize the performance of data-driven applications in SDN-aware distributed systems based on different objective requirements?* The question is categorized as an NP-hard problem, where NP stands for "Non-deterministic Polynomial time." To solve this question, optimization approaches are required, which are determined to find feasible solutions with single or multiple objectives that maximize and/or minimize variables related to performance metrics, taking into account limited infrastructure resources.

1.1.1 NP-Hard: data-driven applications in SDN-aware cloud datacenters

Each data-driven application (e.g., MapReduce [30]) mainly relies on computing, storage, and network components and requires different configurations for each component. Considering a simple scheduling problem where MapReduce tasks are mapped into appropriate cloud virtual machine (VMs) – such that the number of used VMs is minimized and each task of MapReduce is served. The objective here is to minimize cloud energy consumption while meeting MapReduce execution requirements. To prove this problem belongs to an NP-hard class, it can be transformed into a bin-packing problem [31] – that is, the mappers of a given MapReduce application are represented as objects, whereas VMs are represented as bins. Such transformation maps the bin-packing problem into the simplest MapReduce scheduling problem. Thus, this simple problem is proven to be at least as hard as a bin-packing problem, which is already proved to be NP-hard [31]. Adding network requirements to the MapReduce scheduling problem (e.g., an optimal path between MapReduce distributed tasks using SDN) would require cross-layer optimization, which makes the problem a strong NP-hard. To study and solve such kind of NP-hard problems, new modeling and simulation abstractions of MapReduce running in SDN-aware cloud datacenters are required.

1.1.2 NP-Hard: data-driven WAN traffic between distributed SDN-aware cloud datacenters

Several companies have leveraged SD-WAN to manage and control the network connections and traffic between geographically distributed datacenters, such as Google's SD-WAN [16], Microsoft SD-WAN [17], and Facebook Express Backbone[1]. Some problems these companies encounter are how to determine the required number of SDN controllers, how to determine the placement of SDN controllers, how to obtain best routes for data-driven applications in real-time, to name a few. Such problems can be categorized as optimization problems [32, 33]. The placement of the SDN controllers, as an example, is categorized as an NP-hard problem [32, 33]. Still, studying NP-hard problems and evaluating new proposed solutions in the context of multiple SDN-aware cloud datacenters interconnected via SD-WAN networks require new modeling and simulation abstractions.

1.1.3 NP-Hard: data-driven SD-WAN traffic between distributed SDN-aware edge-cloud datacenters

Several IoT-based data-driven applications require to process data across edge-cloud infrastructures. For example, online cloud games (e.g., First-Person Shooter) continuously produce massive amount of data that are required to be processed across edge-cloud systems. Such applications are very sensitive to delay, which require low-latency data transmission and processing [34]. The study in [35] concludes that the maximum tolerable delay of online games should be less than 100 milliseconds (MS) in order to obtain a reasonable gaming experience. To improve players' Quality of Experience (QoE), such as minimizing jitter time (inconsistency of delay rates) [36], optimal deployment decisions for data computations across edge-cloud are required, in addition to the leveraging of the SDN and SD-WAN paradigms. The deployment of every online gaming application is very complex as it requires to distribute computations across SDN-aware edge-cloud environments based on multi-objective criteria and constraints, such as minimizing network latency, minimizing processing time, and

[1] https://engineering.fb.com/2017/05/01/data-center-engineering/building-express-backbone-facebook-s-new-long-haul-network/

prioritizing SD-WAN traffic. Such deployment and configuration problems can be categorized as strong NP-hard optimization problems. To evaluate the strengths and weaknesses of new optimization solutions in the context of edge-cloud infrastructure and end-to-end SDN-aware, new modeling and simulation that captures the abstractions of such ecosystems are required.

1.1.4 Evaluating SDN-related NP-hard solutions

Solving NP-hard problems are intractable because the more inputs are given to a solution's function, the more time the solution's function takes to find an answer [32, 33]. For example, finding a feasible solution for the aforementioned simple MapReduce scheduling problem usually requires a massive search of all candidate solutions of possibilities and combinations. Unfortunately, there is no known way to find a solution that solves NP-hard problems quickly (in a polynomial time) [37, 38]. However, several techniques are used to find an approximation answer for a given problem, subject to the problem's objectives and constraints. For example, heuristic algorithms along with approximation algorithms can be used to reduce a solution's search space, which often leads to an optimal or sub-optimal solution [3, 39].

The behavior and impacts of proposed SDN-aware solutions under various hypotheses must be evaluated to ensure their practicality. One evaluation approach is to test the solutions in real computing environments. However, real SDN-aware environments might be inaccurate due to the behavior instability of computing and network infrastructures. They also impose several challenges, such as high cost, time-consuming, and configuration complexity [1]. Moreover, every application might require several QoS requirements across computing and network infrastructures, which would make the evaluation process very difficult to achieve.

An alternative evaluation approach is to use simulation-based tools [40]. Simulation tools seamlessly offer unified, generic, and customizable knowledge representation models. They can capture domain-specific information required for modeling the behavior of hardware and software components relevant to a given environment. Besides, they offer the ability to simulate large-scale infrastructures in an easy, repeatable, controllable, and configurable manner. Most importantly, simulation tools undertake

various "what-if" investigation and analysis, which aims to measure the impact of changing/tuning the values of dependent and/or independent variables on complex ecosystems (e.g., SDN-aware cloud, SD-WAN).

1.2 Research Challenges

Developing novel modeling and simulation techniques for several SDN-aware environments in order to study and test optimization solutions are full of new research challenges. For every environment (e.g., SDN-aware clouds running MapReduce applications), there must be unique modeling approaches. The following illustrate some of the challenges considered in this thesis:

- **Infrastructure heterogeneity:** The modeling and simulation requires the use of a multi-layered architecture (applications, servers, and networks). Each layer constantly evolves with heterogeneous components, data workflow, and protocols, which might involve a number of different behaviours and configuration parameters. Moreover, every SDN-aware environment (e.g., edge, cloud) requires unique modeling characteristics in order to capture the underlying system abstractions.

- **Coordination complexity:** The modeling and simulation of coordination mechanisms require the components of every environment to communicate, direct, and possibly control other components in terms of data flow and processing along with requesting for specific actions to be undertaken. For example, components from different computing environments (e.g., SDN cloud controllers, SD-WAN controllers) require to coordinate with one another in order to forward network traffic to destinations.

- **Network complexity:** As every SDN-aware environment has its own network layer, it is required to model such networks in dynamic manners using graph theory modeling approaches. Such modeling is essential in order to avoid the hard coding of network topologies, which definitely decreases the practicality of SDN-aware simulation tools. The modeling of network communication would

also be required in order to simulate different wireless and wired network mechanisms and protocols. Further, the modeling of complex control and data flow dependencies within and between different environments are required in order to perform highly detailed SDN-aware simulations.

- **Complexity of application graph:** Increasingly, applications can be represented as a graph of distributed components, tasks, and services, with different data and control flow dependencies between them encoded in the graph. An application can be executed across multi-SDN environments with heterogeneous sets of QoS criteria. Such graph complexity needs to be modelled in a manner that enables different SDN-aware system models to co-exist and execute given applications properly.

- **Policy modeling:** Each application can have specific functional and QoS requirements that needs to be enforced into systems' layers. For example, a MapReduce application might require to find a set of VMs that minimize task execution time in addition to finding optimal routes that minimize data transmission time between tasks. Such requirements must be modeled in a way that it can be easily translated and submitted to given SDN-aware simulation tools.

- **Validation and evaluation:** Validating the generalizability of proposed models, techniques, and abstractions within SDN-aware simulation tools is required in order to illustrate the level of accuracy of the tools as compared to real-world SDN-aware environments. Moreover, validation and evaluation is crucial in order to test efficacy and prove that proposed models are capable of producing results that are realistic and reflective of existing SDN-aware ecosystems.

1.3 Research Aims

As a response to these motivations and challenges, this PhD thesis aims to introduce an extendable toolchain that delivers novel simulation and modeling of SDN-aware environments. It investigates, develops, and proposes novel techniques and frameworks for modeling several SDN-aware environments. The thesis also aims to introduce a

number of optimization solutions in several SDN-aware environments. The thesis's overall outcome is to provide multi-purpose SDN-aware simulations, which enable researchers to evaluate and analyze their proposed solutions and hypotheses with less effort, time, and cost.

1.4 Research Questions

The main question in this PhD research can be stated as *how to optimize the performance of data-driven applications in SDN-aware distributed systems based on different objective requirements?* To answer this question, this PhD research divides the question into four sub-questions. As such, this thesis is guided by the following four questions:

1. How to abstract, model, and represent SDN-aware networks and MapReduce big data applications in cloud datacenters, considering the heterogeneous sets of criteria and components of SDN, MapReduce, and cloud infrastructures? As the performance of MapReduce applications require joint-optimization of host and network layers in the context of SDN-aware cloud datacenters, it is required to investigate and capture underlying components and factors that have high impacts on MapReduce applications running in SDN-aware cloud datacenters.

2. How to abstract, model, and represent SD-WAN networks that interconnect distributed SDN-aware cloud infrastructures considering the complexity of SD-WAN and SDN altogether? Distributed cloud datacenters require massive amounts of data to be transferred to one another via SD-WAN (e.g., replicating data across datacenters for high availability). As such, new optimization solutions are essential to ensure optimal performance and best practices for resource utilization in a global manner. Still, new proposed optimization solutions need to be tested against a range of probable outcomes in order to ensure their practicality; therefore, the outcome of this question would provide a testing framework based on proper capturing of SD-WAN and SDN-aware cloud systems.

3. How to efficiently execute the workflows of data-driven applications across multi-

ple SD-WAN-enabled cloud datacenters and at the same time minimize non-green energy consumption and cost while avoiding SLA violations? Such workflows often lead to an enormous increase in energy usage which is not only a financial burden but also increases overall carbon footprints. To solve such problem, new system and algorithmic solutions are required, considering several factors, such as the application topology and deadline of data-driven applications, the usage of green energy, leveraging of SD-WAN capabilities, and decreasing overall energy consumption and cost.

4. How to abstract and represent different models of SDN-aware networks running in different types of datacenters, including large-scale cloud datacenters and small-scale edge datacenters, in addition to IoT and SD-WAN ecosystems? Interconnecting the layer of IoT data producers to several hierarchy layers of systems (known as osmotic computing), as previously shown in Figure 1.1, requires the gluing and interaction of multiple complex models. This type of emerging ecosystem introduces an unprecedented scale of NP-hard problems. To properly evaluate the strengths and weaknesses of new osmotic-based solutions, a framework with different types of SDN-aware models is required.

1.5 Research Contributions

As SDN and SD-WAN can be involved in several domains, this thesis focuses on three types of ecosystems: (a) SDN-aware cloud datacenters running MapReduce big data applications, (b) distributed SDN-aware cloud datacenters interconnecting via SD-WAN, and (c) SDN-aware osmotic computing. The main contributions of this thesis are listed below:

- To solve the first discussed question, this thesis proposes a new holistic framework that simulates MapReduce applications, big data management systems (BDMS), and SDN-aware networks in cloud-based environments (demonstrated later in Chapter 3). Theoretical and mathematical models of MapReduce in SDN-aware cloud datacenters are also proposed. It also proposes multiple system models to simulate different samples of MapReduce running in SDN-aware

cloud infrastructures. The framework's accuracy is validated against a real-world MapReduce SDN-aware infrastructure.

- To solve the second discussed question, this thesis proposes a novel framework that simulates and models SD-WAN and distributed SDN-aware datacenters (demonstrated later in Chapter 4). It provides accurate modeling of TCP and UDP protocols in addition to network delays. A new SD-WAN routing technique is proposed based on graph theory, which dynamically computes the best route for every network flow. A new scheme for SDN and SD-WAN controllers is proposed to coordinate network functionalities in global sense. The framework's accuracy is validated against a real-world SDN-aware infrastructure.

- To solve the third discussed question, this thesis proposes a new SD-WAN based Workflow Broker (SDWAN-WB) that deploys data-driven workflows across multiple SD-WAN-enabled datacenters, in addition to automating resource provisioning, task provisioning, and data provisioning (demonstrated later in Chapter 5). An adaptive Genetic Algorithm (GA) is also proposed for selecting near-optimal solutions based on green energy usage, the topology and deadline of data-driven workflows, and overall energy consumption and cost. The question is also solved by leveraging SD-WAN capabilities to decrease overall network energy consumption. Extensive experimental evaluation compared with other baseline algorithms is conducted to study the feasibility of the proposed scheduling algorithm and system. This was a collaborative work in which I precisely contribute in network and energy modeling as well as the implementation of the proposed system and algorithm in our proposed IoTSim-SDWAN simulator , as demonstrated in Chapter 4.

- To solve the fourth discussed question, this thesis proposes a unified osmotic computing framework that models and simulates complex IoT applications running over heterogeneous edge-cloud environments while interconnecting via SD-WAN infrastructures (demonstrated later in Chapter 6). Theoretical and mathematical models of osmotic computing are also proposed, in addition to several system management policies. A case study validation using an energy management and

billing application is presented, highlighting the unique capabilities provided by IoTSim-Osmosis for analyzing various parameters, such as IoT energy usage, execution time, and network transmission delays.

1.6 Thesis Structure

The structure of the thesis is shown in Figure 1.2. The chapters are derived from several articles which were published during the PhD candidature. The rest of the thesis is organized as follows:

- Chapter 2 illustrates the background of SDN-aware environments and applicationsand discusses different techniques for modeling and simulation.
- Chapter 3 presents a modeling and simulation framework of big data management system, MapReduce programming model, and SDN-aware networks in a cloud computing environment. This chapter is derived from:
  - **K. Alwasel**, R. N. Calheiros, S. K. Garg, R. Buyya, M. Pathan, D. Georgakopoulos, and R. Ranjan, "BigDataSDNSim: A Simulator for Analyzing Big Data Applications in Software-Defined Cloud Datacenters," *Journal of Software: Practice and Experience*, October, 2020.
- Chapter 4 introduces a framework that models and simulates SD-WAN and distributed SDN-aware cloud datacenters. This chapter is derived from:
  - **K. Alwasel**, D. Jha, E. Hernandez, D. Putha, M. Barika, B. Varghese, S. Garg, P. James, A. Zomaya, G. Morgan, and R. Ranjan, "IoTSim-SDWAN: A Simulation Framework for Interconnecting Distributed Datacenters over Software-Defined Wide Area Network (SD-WAN)," *Journal of Parallel and Distributed Computing*, May, 2020.
- Chapter 5 proposes an adaptive solution for scheduling the workflows of data-driven applications across multiple SD-WAN-enabled cloud datacenters. This chapter is derived from:

  - Z. Wen, S. Garg, G. Aujla, **K. Alwasel**, D. Puthal, S. Dustdar, A. Zomaya, and R. Rajan, "Running Industrial Workflow Applications in a Software-defined Multi-Cloud Environment using Green Energy Aware Scheduling Algorithm," *IEEE Transactions on Industrial Informatics*, December, 2020.

- Chapter 6 presents a framework that models and simulates multiple osmotic systems in a unified environment, which enables the integration and communication of IoT, edge and cloud ecosystems via an SD-WAN and SDN networking. This chapter is derived from:

  - **K. Alwasel**, D. Jha, F. Habeeb, U Demirbaga, O. Rana, T. Baker, S. Dustdar, M. Villari, P. James, E. Solaiman, and R. Ranjan, "IoTSim-Osmosis: A Framework for modeling and Simulating IoT Applications over an Edge-Cloud Continuum," *Journal of Systems Architecture*, November, 2020.

- Chapter 7 summarizes the thesis and illustrates directions for future work.

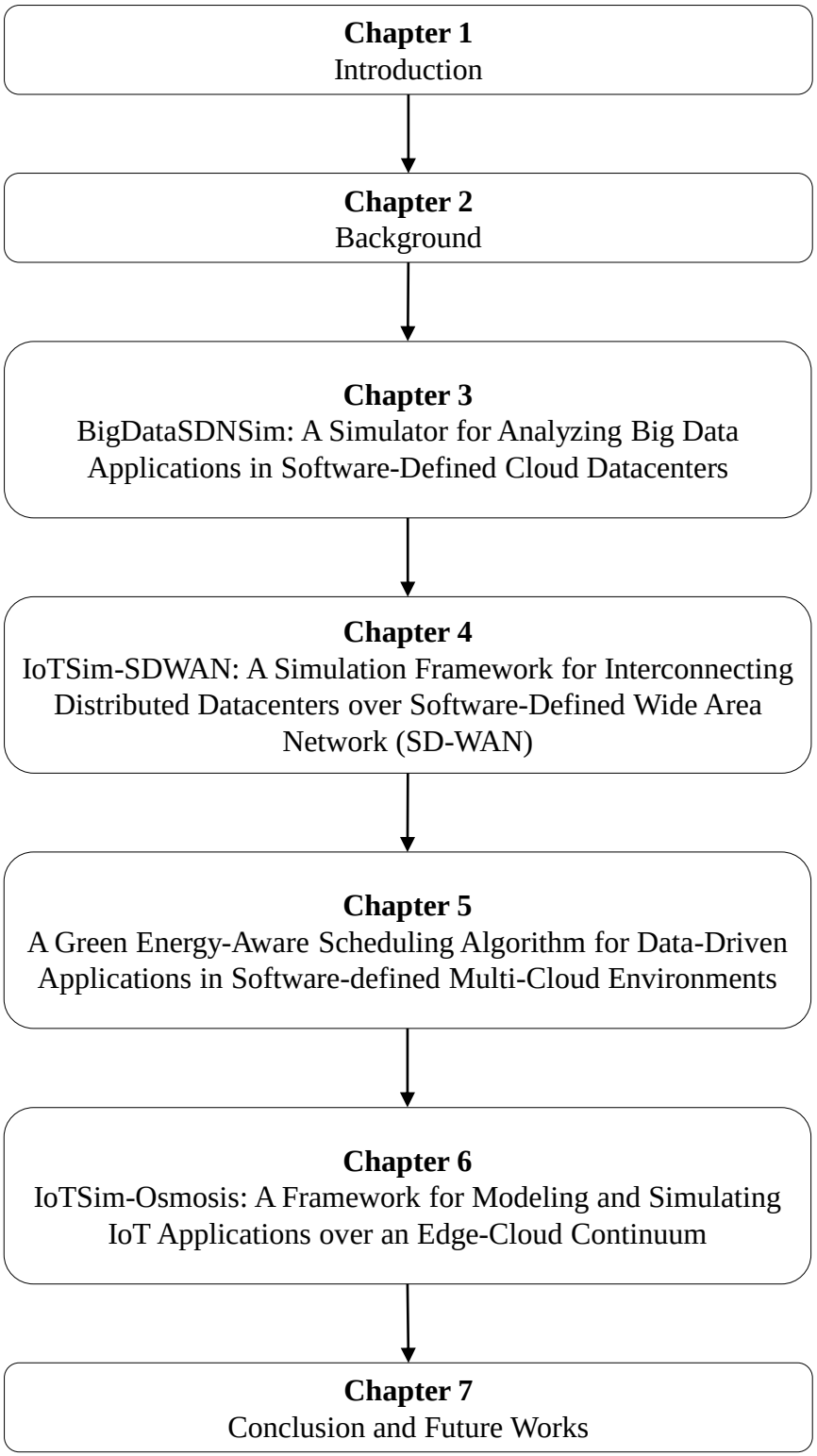


Figure 1.2: Thesis organization

2

Background

Contents

Summary

This chapter demonstrates background and relevant concepts of the overall topics addressed in this thesis. It starts by describing SDN and its architecture as compared to traditional networks. Next, it discusses the deployment of SDN in several environments. In particular, it illustrates the deployment of SDN in cloud datacenters to enhance the overall performance of MapReduce applications. Next, it demonstrates the leveraging of SD-WAN between several distributed cloud datacenters. It also discusses the use of SDN within IoT edge datacenters. In addition, this chapter describes how SDN-aware solutions can be evaluated.

2.1 Software-Defined Networking (SDN)

SDN is a networking paradigm originally derived from the work of Martin Casado in 2005 [41]. SDN was initially designed to simplify the process of network management and configuration. The rising of SDN started with the invention of the OpenFlow (OF) SDN protocol in 2008 [42]. OF is a network management protocol to control the flows of network packets in a given SDN-aware network. The main pillars of SDN include:

- Moving network control logic from network devices (e.g., routers, switches) to SDN controllers.
- Real-time programming of networks through software applications and/or application programming interfaces (APIs).
- Global view of data flow across given networks.
- Making network decisions (forwarding and routing) based on different criteria, such as applications, flows, and Internet Protocol addresses (IPs).

SDN aims to enable dynamic configuration of networks and overcome the limitations of traditional network infrastructure [9]. The key difference between traditional networks and SDN is that the control layer of network devices, which acts like a ”brain,” is moved

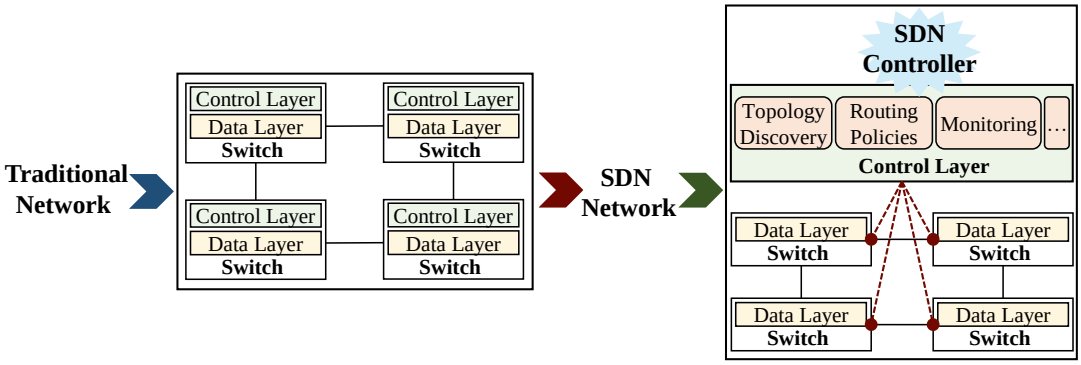


Figure 2.1: Key difference between traditional networks and SDN-aware networks

away to an SDN controller, as shown in Figure 2.1. Such a tactic makes networks more robust, simplified, and flexible to changes as the network is controlled from a central point instead of complex, distributed control mechanisms. The control layer seamlessly enforces SDN customized management and routing policies in the data layer using well-defined APIs, such as OpenFlow [42]. As a result of the attractive features and world-wide adoption of SDN, several network vendors (e.g., HP, Pica8, Netgear) have introduced numerous network devices that are SDN-aware, which bring a better consumer experience.

2.1.1 SDN architecture

Figure 2.2 demonstrates an overview of SDN architecture and OF. The architecture consists of three layers: data, control, and management. The data layer contains network devices that implement an OF protocol. It allows remote management and access to heterogeneous devices without exposing internal designs and functionalities. Every network device maintains an OF table in order to enable an SDN controller to configure its network states dynamically. For example, a routing table embedded in a given network device can be used to add, retrieve, remove, and update routing entries on behalf of a respective SDN network manipulation functions and applications (e.g., routing, monitoring, traffic load-balancing).

The control plane contains one or multiple SDN controllers where they expose several southbound and northbound APIs to the data and management layers, respectively. The data layer enables SDN controllers to instruct and manipulate network devices' rules seamlessly via an OF protocol and the southbound APIs. On the other hand, the

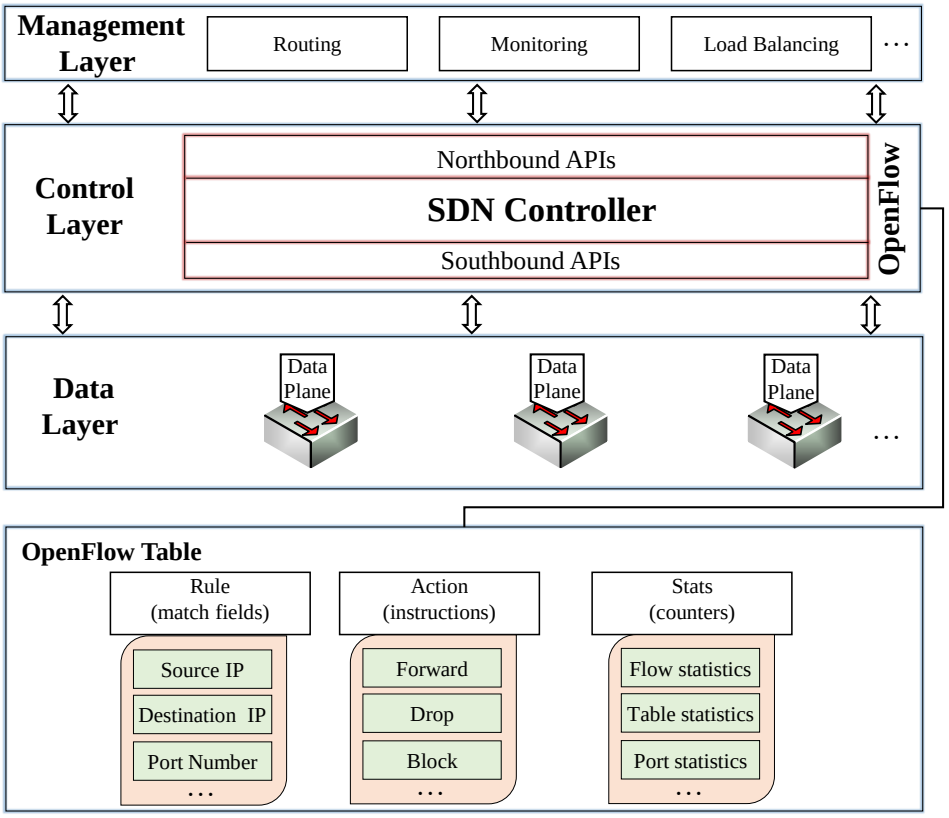


Figure 2.2: SDN and OpenFlow overview

management layer enables SDN controllers to deploy the functions of various network applications via the northbound APIs. Such network applications are used to configure, control, and orchestrate an entire given network without having direct communications with network devices. Every application can contain various parameters and functions which are injected into SDN controllers.

2.2 SDN deployment

Several domains have adopted SDN at a rapid pace as it offers promising performance and management solutions. SDN has re-moulded the long-established, traditional networks to a smarter and innovative infrastructure in order to keep pace with the digital age's rapid development. In particular, SDN networking revolution empowers the new era of cloud computing, data-driven applications, edge computing, IoT, and SD-WAN.

2.2.1 Cloud computing

Cloud computing is a novel way of delivering and offering on-demand Information technology (IT) infrastructures and services via virtualized utilities [43]. It is based on subscription and consumption models where consumers pay for services based on their usage. It consists of several geographically distributed datacenters where each datacenter has its own components, such as storage, networking, and computing. Cloud consumers can seamlessly scale up and down their computing requirements (e.g., CPUs, VMs) according to a Service Level Agreement (SLA).

Cloud computing offers three services: Software as a Service(SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [44]. SaaS provides off-the-shelf software applications that are ready for immediate use. Such services are beneficial for start-up companies to accelerate the launching phase along with streamlining operations. PaaS plays a vital role in building customized applications with particular software elements. It supports adding special services and properties to applications while a trusted third party manages underlying IT infrastructures. IaaS supports the idea of managing an underlying IT infrastructure through virtualization mechanisms. It fulfills consumer needs by providing complete control over resources, such as VMs, storage, and services.

2.2.2 Big Data

Big data analytics has emerged as the preferred option to effectively analyze large-scale data sets at an astonishing speed as compared to traditional database systems, which require excessive time and a single super-computing capability [45]. Big data harnesses the power of clustering commodity hardware to carry out data analysis in a parallel and simultaneous manner. It has become the most prominent mechanism to delve into data sets and provide valuable insights, such as detection of emerging risks and threats, predicting behaviors and patterns, and providing business opportunities [46]. In general, data is defined as ”big data” if they share three characteristics, which are variety (various data types), volume (a massive among of data), and velocity (data processing speed) [12].

To bring big data analytics into existence, several frameworks have been developed and launched in cloud-based environments, such as Hadoop MapReduce [30] and Apache Storm[47]. These frameworks facilitate the process of utilizing parallel programming models and engines, along with meeting different aspects of big data requirements. For instance, the MapReduce programming model can be used to analyze historical data, while a stream processing model is used to handle a never-ending data stream at a speed of milliseconds [48]. Moreover, a big data application can rely on multiple big data models and engines (e.g., Apache Hadoop YARN[49]) to handle different aspects of data analysis simultaneously. As such, big data management systems (BDMS), YARN, have emerged to coordinate and schedule resources among big data engines and applications co-hosted on a shared big data cluster.

2.2.3 WAN to SD-WAN

WANs are the core communication infrastructures that interconnect geographically distributed systems and devices into a single network [50]. Although traditional WANs have been developed to interconnect distributed systems, there are some limitations with respect to lack of adaptive routing behavior, unbalanced load distribution, requirement of complex network protocols, lack of prioritization and the need for specialist hardware. Due to these drawbacks, the management and deployment of traditional WANs in the context of data-driven applications is limited. For a complete distributed cloud datacenters that offers better resource management and efficiency within modern data-driven applications (e.g., smart energy clouds, content delivery network, distributed gaming), the WAN needs to be part of the whole adaptable SDN solution.

SD-WAN originates from the SDN paradigm which leverages the SDN mechanisms of managing, operating, automating, and simplifying networks within a WAN context [51]. SD-WAN has emerged as a promising solution to alleviate the issues of classical WAN along with the objectives to enhance the performance and deployment of various data-driven applications [16, 17, 52]. The concept of SDN's decoupling of the control and data layers is applied to the SD-WAN ecosystem, in addition to the leveraging of software-based centralized controllers. SDN is primarily responsible for controlling

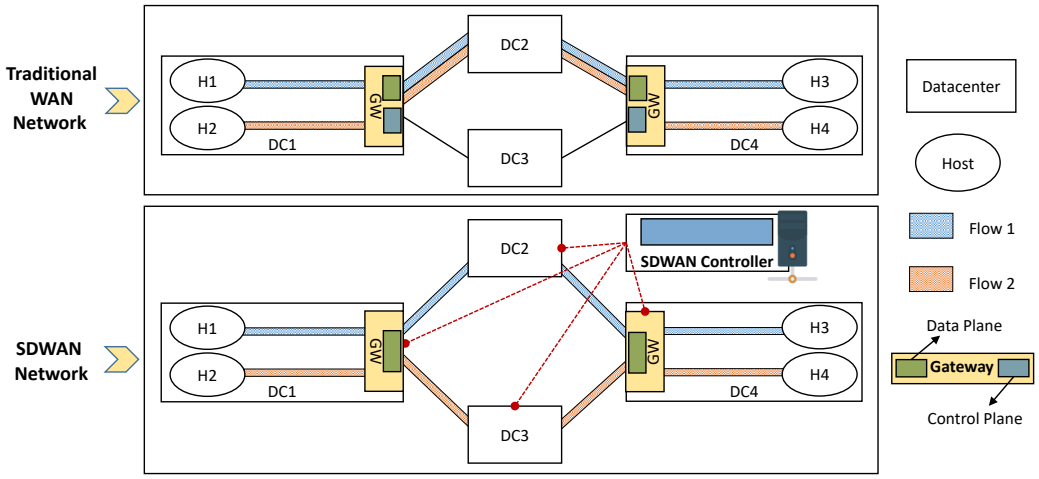


Figure 2.3: Key difference between classical WAN and SD-WAN networks

and managing datacenters' internal network operations (e.g., cloud and edge), whereas SD-WAN moves the focus to manage the interconnecting geographically distributed applications, systems, and datacenters.

The earliest integration of an SD-WAN ecosystem for improving network utilization was by Microsoft [17] in 2013, followed by Google in 2014 [16]. Both Microsoft and Google leverage SD-WAN solutions to accelerate the process of copying a large amount of data across and between datacenters while improving network performance, coordination, traffic engineering, and overall resource optimizations. Recently, cloud providers, such as Amazon[1], have leveraged SD-WAN solutions to move from IP-based WAN routing and management mechanisms to application-based SD-WAN ecosystems.

Figure 2.3 illustrates the key difference between WAN and SD-WAN environments. In the classical WAN, a gateway contains both data and control planes, whereas an SD-WAN separates control planes from gateways to a centralized SD-WAN controller. The data plane maintains routing information while the control plane is responsible for making all network decisions and providing the data plane with routing information. From the figure, it can be seen that the SD-WAN controller oversees and manages the whole network. This results in cohesive global network decisions made with an awareness of current (and legacy) traffic issues. The SD-WAN can also enforce a new network policy/QoS on the fly, something that a classical WAN may never achieve due

[1]https://aws.amazon.com/blogs/apn/exploring-architectures-with-cisco-sd-wan-and-aws-transit-gateway/

to its static nature. Moreover, an SD-WAN is capable of load balancing the network in a global sense, by directing data flows across the least congested routes of a network. Such traffic engineering techniques improve network transmission time and the QoS of traffic flow (including improving resource utilization).

2.2.4 Edge computing and IoT

Edge computing is a new computing paradigm with the aim to process and store data closer to the data origin location [53]. It improves the overall performance of distributed data-driven applications by minimizing response time. It also saves network consumption and avoids network choke points by reducing the amount of data forwarded to geographically distributed cloud datacenters. Edge computing can have different environment forms, from a smartphone, to a Raspberry Pi, to a laptop, to small-scale datacenters. Edge datacenters are similar to cloud datacenters with a small number of components, such as servers and switches.

IoT can be described as a set of physical things "devices" with the ability to sense surrounding environments and process sensed data for observing particular patterns [53]. IoT devices can communicate with each other and with respective computing environments based on given configurations. IoT is anticipated to permeate society according to research conducted by Gartner[2] (an IT service management company). Gartner predicts that IoT will connect up to 20.4 billion devices by 2020, which will add around 3 trillion US dollars to the world economy. As a result, IoT has become the backbone of several emerging applications, such as smart buildings, smart cities, smart vehicles, environmental sensing and forecasting, and disaster management, among others [54].

An IoT application, such as a smart meter application, can distribute its real-time computation process across IoT devices, edge and cloud datacenters based on the desired functional and QoS parameters [55]. To obtain such distributed computations, generated data are forwarded from IoT device to edge and further from edge to cloud using different communication and network protocols. The computational result can be used to make some decisive actions to fulfill the desired application process.

[2]http://www.gartner.com/newsroom/id/3598917

2.2.5 Osmotic computing

The 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 datacentres [56]. 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 a 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, which can be defined as *osmotic computing* [25].

The osmotic computing paradigm provides a simplified model for the deployment of IoT applications in the integrated edge-cloud environment [25]. It focuses on the design and implementation of a unified computing model that leverages the capabilities of various distributed systems and networks (IoT-edge, cloud datacenters, and SD-WAN). Osmotic computing suggests the migration of workload to/from a cloud datacentre to edge devices via SD-WAN, based on performance and security *trigger* events. The aim of osmotic computing 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.

2.3 Evaluation methods

The evaluation of proposed SDN-aware designs, architectures, and algorithms in a given environment (cloud, edge, etc.) is essential to estimate the level of performance considering a wide-spectrum of systems' behaviors and components. In general, the SDN evaluation process can be conducted using three different methodologies: experiments with real SDN-aware environments, SDN-aware emulation, and SDN-aware simulation.

2.3.1 Real SDN-aware environments

Leveraging real SDN-aware environments is an option for evaluating new SDN-aware solutions. Such environments are often conducted in small-scale infrastructures, which

might understate or overstate the effectiveness of proposed solutions [1]. Also, most scientists and researchers cannot obtain such infrastructures due to several reasons related to financial and technical constraints. Albeit live production environments are feasible, it is not recommended to be used during the testing phase because proposed solutions might lead to failure or performance degradation [57]. Moreover, to carry out real SDN-aware environments, switches and routers need to be SDN-aware with an embedded OF protocol. An SDN controller software (e.g., Ryu[3], OpenDaylight[4]) is also required to control and configure network devices via an OF protocol.

2.3.2 SDN-aware emulation

An emulator can be defined as a hardware or software tool designed to imitate similar functions of given hardware platforms, operating systems, or applications [58]. With the rapid development of technology, emulation-based tools and strategies have become essential to accelerate the development of a given IT environment vastly. For example, SwiftUI[5] and Android Studio[6] are emulators designed to enable developers to create new smartphone applications using computers without the need to obtain real smartphone devices and operating systems. Similarly, several emulators exist to imitate SDN-aware environments. For instance, Mininet[7] is a network emulator designed to create a realistic virtual SDN-aware network where the network is controlled by an SDN controller (e.g., Ryu[8]). Although emulators are powerful tools to be used based on given requirements, SDN-aware emulators suffer from several shortcomings, such as the inability to emulate distributed cloud datacenters where each datacenter has its network and computing infrastructures [59].

2.3.3 SDN-aware simulation

Modeling can be defined as an abstract representation of real-world systems in which it captures functions and properties [60]. On the other hand, a simulation is an act

[3]https://ryu-sdn.org/
[4]https://www.opendaylight.org/
[5]https://developer.apple.com/xcode/swiftui/
[6]https://developer.android.com/studio
[7]http://mininet.org/
[8]https://ryu-sdn.org/

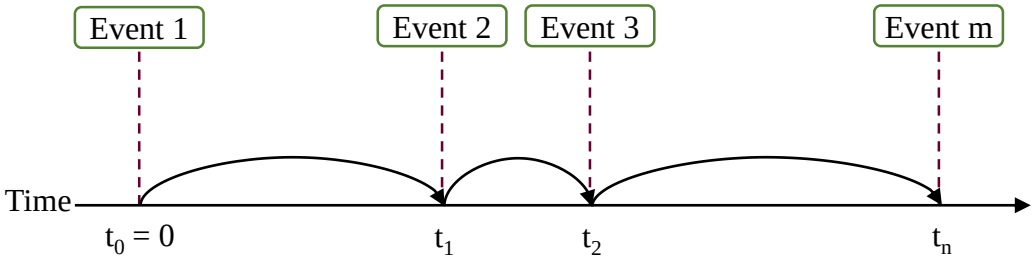


Figure 2.4: Discrete-event overview

of executing models in a code-base programming manner [60]. Compared with real-world systems and emulators, simulation is capable of simulating large-scale distributed infrastructures without much cost and in a configurable and easy manner [1, 60]. It is the most practical approach to quantify new solutions with acceptable accuracy. Simulation tools might rely on one or several models to effectively capture underlying system behaviors, depending on the system's complexity. In general, simulation tools can be designed in two approaches: continuous or discrete-event [60]. Also, simulation tools can be hybrid in which they combine the two approaches [60].

2.3.3.1 Continuous simulation

A continuous simulation is an act of modeling the properties and behaviors of real-world objects in continuous functions where a set of dependant variables are continuously changing with respect to time [60]. It is typically based on differential equations to demonstrate the rates of change in a given system's states [60]. For example, vehicles' motion can be simulated based on a continuous approach where the states of motion continuously change. While a continuous approach is capable of tracking an exact motion, a discrete-event approach can be used as an alternative to provide an approximation of vehicles' motion.

2.3.3.2 Discrete-event simulation

A discrete-event approach simulates the change of state of real-world objects as a set of events occurring at particular instances of time [60]. It is commonly used in several domains, such as health care, logistics, transportation, manufacturing, and computing [61, 62]. Figure 2.4 illustrates the principle of the discrete-event approach, where events are intercepted with respect to time. The activities occurring between

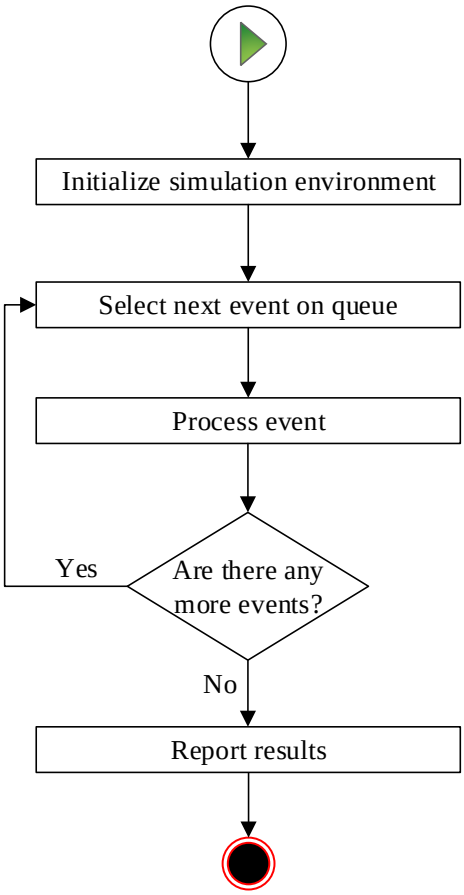


Figure 2.5: Discrete-event model

the events are neglected as they do not change systems' behavior as compared to the continuous approach.

The main components of a discrete-event simulation environment are entities, a clock, and an event queue. The entities represent stand-alone elements with their unique functions, procedures, and behaviors, such as SDN controllers, switches, and servers. Each element can be mapped into several different objects (e.g., numerous SDN controllers) where each object can have its own functions and data structures. The simulation clock is continuously updated according to the time of every intercepted event. In this case, a simulator can be described as a scheduler where it executes events in a sequential manner. Once there are no more events in the event queue, simulation will finish and statistics and results will be reported.

3

BigDataSDNSim: A Simulator for Analyzing Big Data Applications in Software-Defined Cloud Datacenters

Contents

Summary

To support the testing and bench-marking of MapReduce applications that rely on data processing and transmission across multiple VMs, this chapter presents a new, self-contained simulation tool named BigDataSDNSim. The new simulator enables the modeling and simulation of the big data management system (YARN), its related programming models (MapReduce), and SDN-aware networks in a cloud computing environment. The chapter also validates and compares the accuracy and correctness of BigDataSDNSim with a real MapReduce SDN-aware environment. Finally, the chapter presents two uses cases of BigDataSDNSim, which exhibit its practicality and features, illustrate the impact of data replication mechanisms of MapReduce in YARN, and show the superiority of SDN over traditional networks to improve the performance of MapReduce applications.

3.1 Introduction

The advent of the MapReduce programming model, BDMS, and clouds has contributed to improving the existing practice of data analysis and synthesis. However, one critical issue of MapReduce is that every MapReduce application faces several performance challenges due to its unique context and requirements of data flow and processing [48, 51, 63, 64]. In particular, every MapReduce application has its characteristics and runs across tens of servers distributed in different datacenter racks; therefore, every application would most often require unique scheduling mechanisms, subject to the given processing and communication requirements and patterns [12]. Undoubtedly, MapReduce host-network scheduling plays a vital role in achieving performance goals, reducing execution time, minimizing computing costs, and ensuring proper resource utilization and management [65, 66].

The architecture and performance of MapReduce, which is managed by a BDMS, depends on two major factors: processing and network transmission. Most of the existing solutions inform design decisions in terms of MapReduce processing performance and neglects the transmission performance due to the lack of real-time, dynamic network configurations. With the advent of SDN, a number of studies have leveraged

MapReduce in SDN-aware environments and proposed novel joint-optimization solutions, which notably enhance MapReduce performance in terms of processing and transmission [3, 4, 67]. Still, several challenges need to be tackled to leverage and evaluate the benefits of SDN for supporting the network capabilities MapReduce applications in cloud contexts. To fill this gap, we present BigDataSDNSim: a new, discrete-event simulation tool designed to enable the modeling and simulation of big data management systems (YARN), big data programming models (MapReduce), and SDN-aware networks within cloud computing environments.

To the best of our knowledge, BigDataSDNSim is the first tool that models and simulates the three merging technologies (MapReduce-BDMS, SDN, cloud) in a single simulated environment. Based on our proposed system-based and mathematical models, the simulator is capable of capturing the key functions, characteristics, and behaviors of the SDN-aware computing environment. It can also model the functionalities of MapReduce applications in line with mimicking diverse SDN capabilities and interactions with BDMS systems in a seamless manner.

The main contribution of BigDataSDNSim is the ability to generate different samples of SDN-aware MapReduce-BDMS infrastructures, along with reducing the complexity of deploying new SDN-aware MapReduce optimization solutions. The simulator can predict and quantify the impact of new SDN-aware MapReduce solutions and designs running in cloud environments. The accuracy and correctness of BigDataSDNSim are validated by comparing the behavior and results of a real environment that combines MapReduce and SDN with an equivalent simulated environment given by BigDataSDNSim. Evaluation results demonstrate that BigDataSDNSim is closely comparable with real environments. To gain insight of the features and functionalities of BigDataSDNSim, this chapter present two use cases that demonstrate the ability of SDN to improve the performance of MapReduce applications as compared to traditional networks and illustrate the impact of Hadoop Distributed File System (HDFS) replication mechanisms in the overall MapReduce performance. In summary, the main contributions of this chapter are as follows:

- A new holistic simulation framework that simulates MapReduce applications, BDMS, and SDN-related networks in a cloud-based environment

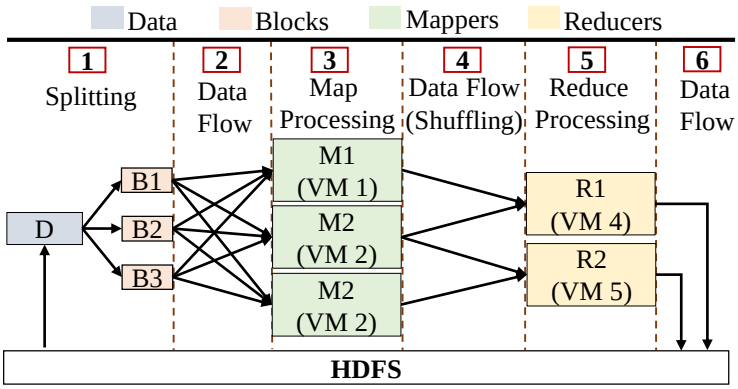


Figure 3.1: An overview model of MapReduce processing and data flow phases

- Theoretical modeling of MapReduce processing and transmission in SDN-aware cloud datacenters
- Multiple system models to simulate different samples of MapReduce running in SDN-aware cloud infrastructures
- Validation and comparison of the accuracy and correctness of BigDataSDNSim with a real-world MapReduce SDN-aware environment

The rest of this chapter is organized as follows: Section 3.2 provides an overview of a MapReduce-BDMS architecture running in an SDN-aware cloud datacenter. Section 3.3 presents related work and reflects the importance and unique capabilities of BigDataSDNSim. Section 3.4 elaborates the architecture and design of BigDataSDNSim framework in detail. Section 3.5 demonstrates the modeling of BigDataSDNSim mathematically and descriptively. Section 3.6 validates and compares the accuracy and correctness of BigDataSDNSim with a real MapReduce SDN-aware environment. Section 3.7 presents two use cases of BigDataSDNSim. Section 3.8 concludes the chapter and highlights some future work.

3.2 Overview

MapReduce is a programming model that runs in the form of a big data application [30]. Apache Hadoop YARN [49] is considered to be the most dominant framework for building MapReduce applications. A fundamental part of the YARN is HDFS,

which is a distributed file storage that distributes data across virtual machines (VMs) in a given big data cluster. It supports a replication mechanism of data sets for fault tolerance purposes, which is set up to three replications by default. On data set submission, HDFS breaks down the data into several small data blocks and then replicate and distribute these blocks into selected slave VMs. HDFS tries to balance the distribution of data blocks among VMs fairly; however, the distribution of a given data block can be later altered if all VMs that contain the data block are busy. By using a data locality technique, YARN copies the functions of map and reduce to all salve VMs so that any VM can be selected to execute a map or reduce function. Whenever a MapReduce application is being executed, YARN would specify it as a MapReduce job, which holds information and records, such as the number of data blocks and the execution time of every map and reduce task. So, a MapReduce application and a MapReduce job are interchangeable. To avoid confusion, we use the "MapReduce application" throughout this article.

MapReduce applications can have different building blocks and characteristics based on processing and data flow requirements. The simplest model of MapReduce is illustrated in Figure 3.1, which consists of six phases, assuming that HDFS, mappers, and reducers reside in different nodes. As it can be seen the HDFS splits the data (D) into several blocks (e.g., B1, B2, B3) where each block is replicated and forwarded to elected VMs via a network layer. For every data block, a single mapper is selected to carry out the processing. Once every mapper completely processes the data, it starts the transmission of the output to respective reducers (e.g., R1, R2) according to the key-value pairs. The key-value mechanism of MapReduce requires reducers to wait for all mappers to finish processing so that the output of reducers is accurate. Once every reducer finishes processing, it transfers its output to the HDFS to be combined and reported. The block replication mechanism is also required for reducers to divide their output into small-scale data blocks and transfer the blocks to other elected VMs based on the replication factor.

Figure 3.2 illustrates an overview of SDN-aware YARN systems running MapReduce applications. In a given datacenter, every big data cluster is managed by a resource manager and have a single HDFS shared by all MapReduce applications. The resource

manager contains a list of worker VMs where each VM is controlled and monitored by an implemented agent (node manager). For every requested MapReduce application, the resource manager creates a new application master and links requested VM resources to the application master. Once the application master is activated, it copies the code-based functions of its map and reduce tasks to all respective VMs so that every VM can be elected to run the map and/or reduce task, as previously mentioned to ensure the principle of data locality. Following that, the execution logic of MapReduce applications follows the six processing and transmission phases, as mentioned earlier.

The description above of MapReduce, BDMS, and SDN should be considered to derive correct simulation models and ensure the accuracy of simulated results. BigDataSDNSim, therefore, is modeled and designed accordingly with the given descriptions. It also provides abstract layers for enforcing new policy-based solutions. For example, the components of a single MapReduce application running in a cloud datacenter might be located on different servers and/or racks due to insufficient resource capacity in a single server. Such distribution logic requires accurate abstractions and modeling in order to properly evaluate optimization solutions for MapReduce running across several distributed servers. In another example, the same MapReduce application might require special QoS requirements (e.g., traffic prioritization, policy-based routing mechanisms) and excessive data transmission on the network layer from one server to another. As SDN-aware networks are capable of meeting such requirements, BigDataSDNSim is modeled to seamlessly provide easy deployments of QoS requirements on behalf of every MapReduce application.

3.3 Related works

Several simulation tools were developed due to the surge in adoption of MapReduce and SDN in cloud-based environments. Some of the tools were developed from the ground up while others were developed on top of existing tools. We believe that there is a clear gap in state-of-the-art simulators in terms of modeling and support for MapReduce applications, BDMS, and SDN in cloud-based infrastructures. This section demonstrates existing simulators in terms of their ability to model and simulate

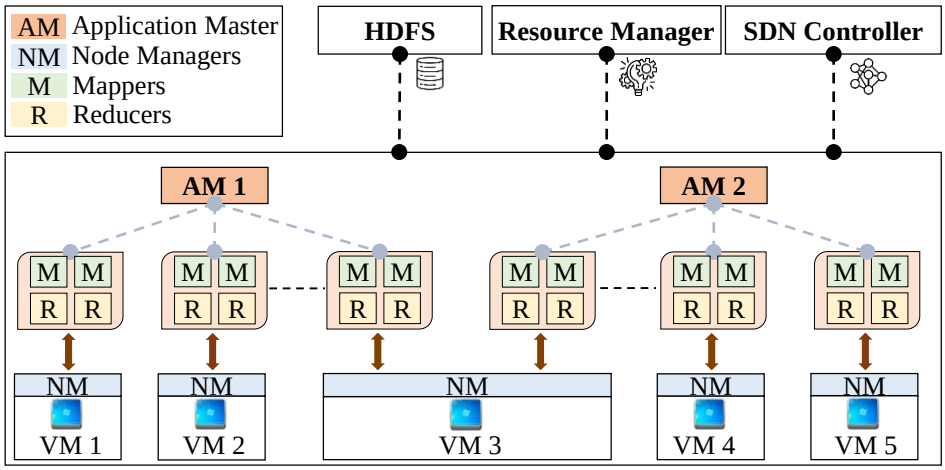


Figure 3.2: Overview of SDN-aware YARN-related systems

MapReduce-DBMS within SDN-aware cloud datacenters. It then illustrates how our simulation framework fulfills the gap of existing simulation frameworks in a holistic manner. Table 3.1 illustrates the differences and similarities of existing simulators and emulators as compared to our simulation framework.

There are several simulation tools capable of simulating and modeling the characteristics of cloud and legacy networks. For example, CloudSim [1] is one of the most popular cloud-based tools that allow the modeling of physical and virtual cloud infrastructures using an event-driven architecture. It is capable of simulating and evaluating the performance of cloud infrastructures as well as deploying various provisioning and allocation policies (e.g., VM placement, CPU task scheduling). CloudSim forms the base upon which several simulation tools were developed to fill the gap of networks and applications aspects (e.g., NetworkCloudSim [68], WorkflowSim [69]). GreenCloud [70], iCanCloud [71], and NetworkCloudSim are other cloud-based tools focusing on the perspective of legacy networks in terms of characteristics and communications in cloud datacenters.

Mininet [72] is an SDN-aware emulation tool that runs on a single device configured with a Linux-based system. It emulates different types of network topologies together with hundreds of virtual hosts and diverse UDP/TCP traffic patterns. The external SDN controller(s) communicates and enforces network policies on Mininet via its unique IP address and OpenFlow APIs. While Mininet is limited to run on a single physical machine, MaxiNet [73] was introduced to enable Mininet to run across

multiple physical machines. MaxiNet is capable of emulating large-scale SDN cloud environments along with evaluating new SDN-aware routing algorithms. Moreover, to allow Mininet to mimic the behaviors of MapReduce applications, MRemu [74] was introduced where it is capable of using realistic MapReduce workloads/traces within Mininet environments. It operates on latency periods extracted from MapReduce job traces (duration of tasks, waiting times, etc.).

Similarly, EstiNet [75] is another SDN-aware simulation and emulation tool, allowing each simulated host to run a real Linux-based system coupled with several types of real applications. It is capable of simulating hundreds of OpenFlow switches and generating real TCP/IP traffic. CloudSimSDN [2] is an SDN-aware cloud simulator that is modeled and implemented on top of CloudSim [1]. It enables the simulation of SDN-network behaviors in a cloud-based environment. The focus of CloudSimSDN is to model power-based management policies to reduce the energy consumption of hosts and network devices. SDNSim [76] is an SDN-aware simulator that simulates datacenter elements (e.g., switches, links, and hosts) along with the layer of SDN data plane. The control plane of SDNSim is handled by an external SDN controller, which enforces network decisions and solutions through APIs. Moreover, SDN-Sim [77] is an SDN-aware simulation and emulation tool that integrates several frameworks to facilitate the end-to-end performance evaluation of wireless technologies (e.g., 5G). The Virtual infrastructure of SDN-Sim depends on VMWare ESXi servers while its network layer is managed by OpenFlow protocol and Opendaylight controller. SDN-Sim's network can be emulated using GNS3 and Mininet. It depends on MATLAB server to run real world SDN wireless scenarios.

IoTSim [78] is an extension of CloudSim that mimics the characteristics of the MapReduce programming model. It allows simulation and modeling of the old version of the Hadoop framework (e.g., job trackers, task trackers). It provides a simple management mechanism to configure MapReduce applications based on IoT-generated data. Similarly, MR-CloudSim [79], MRSim [80], and MRPerf [81] are other tools that enable the simulation of MapReduce-based applications with different focuses and features. In addition, BigDataNetSim [82] is a simulator designed to evaluate the data placement strategies of HDFS within a dynamic network cluster. It focuses on evaluating solu-

Table 3.1: Comparison of related simulators and emulators
✓* with the help of the INET framework [5]; ✓** with the help of SDN controllers from other projects (e.g., Floodlight [6])

Simulators	Features							
	Language	Evaluation Objectives	MapReduce Model	Network Model	BDMS Model	SDN Model	Cloud Model	Dynamic Routing Mechanisms
CloudSim [1]	Java	Performance					✓	
NetworkCloudSim [68]	Java	Performance		Limited			✓	
WorkflowSim [69]	Java	Performance					✓	
GreenCloud [70]	C++/Otcl	Energy		✓			✓	✓
iCanCloud [71]	C++	Performance		✓*			✓	✓*
Mininet [72]	Python	Performance		✓		✓**		✓**
MaxiNet [73]	Python	Performance		✓		✓**	✓	✓**
MRemu [74]	Python	Performance	✓	✓		✓**		✓**
EstiNe [75]	C++ & Python	Performance		✓		✓		✓
CloudSimSDN [2]	Java	Performance & Energy		✓		✓	✓	
SDNSim [76]	MATLAB & Python	Performance		✓		✓	✓	✓
SDN-Sim [77]	MATLAB, Java & Python	Performance		✓		✓		✓
IoTSim [78]	Java	Performance	✓				✓	
MR-CloudSim [79]	Java	Performance	✓				✓	
MRSim [80]	Java	Performance	✓	Limited				
MRPerf [81]	Python	Performance	✓	✓			Limited	
BigDataNetSim [82]	Java	Performance		✓		Limited		✓
MaxHadoop [83]	Python	Performance	✓	✓		✓		✓
BigDataSDNSim (Proposed)	Java	Performance & Energy	✓	✓	✓	✓	✓	✓

tions for transferring HDFS data to distributed nodes while it neglects the logic and dependencies of MapReduce. MaxHadoop [83] is an emulation tool developed on top of MaxiNet [73] to evaluate the performance of MapReduce strategies within an SDN network environment. The network within MaxHadoop is managed by an external SDN controller, such as Floodlight [6].

While these tools are powerful for simulating cloud-based environments, MapReduce applications, traditional networks, and SDN, BigDataSDNSim differs in supporting a holistic simulation framework that simulates MapReduce applications, BDMS, and SDN-related networks in cloud-based environments. In particular, BigDataSDNSim differs in terms of modeling and simulating:

- A generic big data approach for executing different big data programming models (e.g., MapReduce, Stream) simultaneously
- MapReduce applications within big data cluster management (BDMS), which is one of the prominent platforms for running different big data models
- Behaviors and features of SDN dynamic networks coupled with the coordination and interaction with MapReduce applications within cloud environments

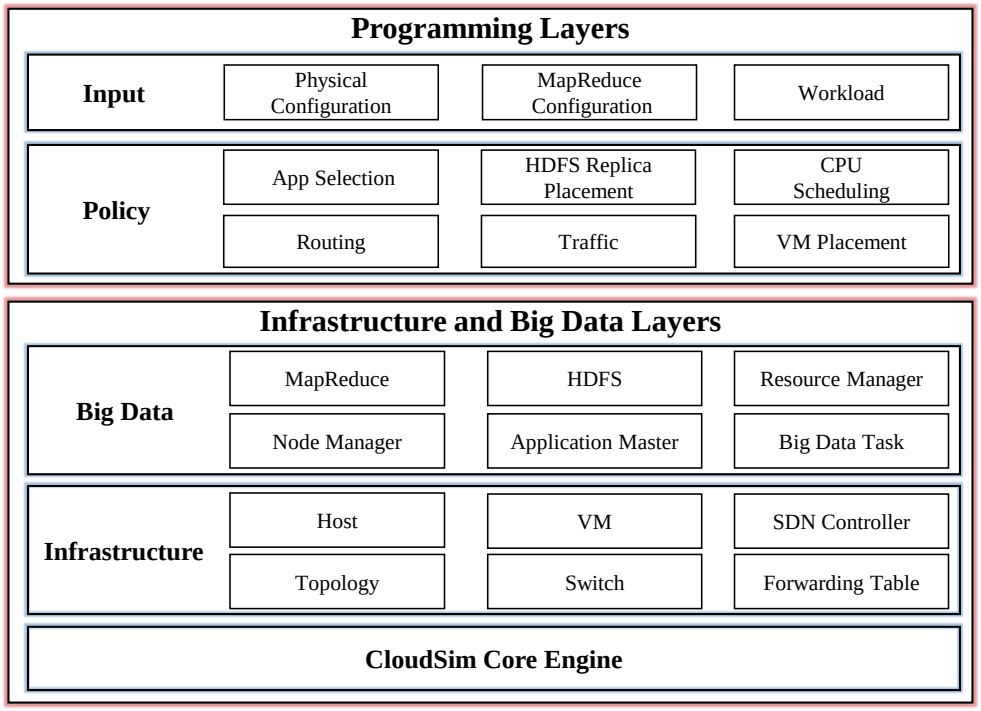


Figure 3.3: Architecture of BigDataSDNSim simulator

- Dynamic routing mechanisms based on graph theory to enable any type of network topology to be seamlessly simulated

- Several policies for SDN, MapReduce, and VM within cloud datacenters for multilevel optimization

3.4 Architecture

This section demonstrates the fundamental functionalities and components of BigDataSDNSim. The modeling logic of our simulation framework is based on the overview in Section 3.2. Figure 3.3 presents the key components of our proposed architecture, in addition to a few used elements of CloudSim and CloudSimSDN. The Figure facilitates our simulator's use by categorizing the architecture into two main layers: programming and infrastructure and big data. The detailed description of every component in every layer is discussed later in Section 3.5. The overall description of each layer is given as follows:

3.4.1 Programming layers

The programming layer facilitates the deployment of strategies, policies, and algorithms. It abstracts away the underlying complexities of BigDataSDNSim. It provides the baseline to implement multi-level MapReduce optimizations in SDN-aware cloud environments. It consists of the following layers:

- *Input*: This layer allows the implementation of different simulation scenarios in a simple manner. The configurations of datacenters (e.g., requirements and descriptions of hosts and networks) can be easily defined using a single JSON file. The layer also provides mechanisms for configuring big data clusters, such as the number of required VMs and MapReduce applications. The descriptions of processing and data transmission of MapReduce applications can be submitted in a CSV file, which includes attributes such as start time and the size of data to be transferred between components (e.g., from HDFS to VMs). By using the input of this layer, BigDataSDNSim instructs its respective components to behave according to the given rules.

- *Policy*: By considering the importance of different policy requirements of MapReduce applications and SDN traffic engineering, this layer is designed to provide a mixed composition of MapReduce and SDN policies. Such policies are key factors for obtaining optimal performance in MapReduce processing and transmission. Therefore, this layer allows the implementation of new algorithms for every listed policy.

3.4.2 Infrastructure and big data layers

The infrastructure and big data layers contain the core, complex components of BigDataSDNSim. If new functionalities emerge for big data programming models and the current SDN capabilities of BigDataSDNSim are not supported, these layers should be extended. In this layer, most of the components communicate with each other using a discrete-event mechanism.

- *Big Data*: This layer holds components responsible for simulating the behaviors and characteristics of MapReduce and BDMS. Big data applications might require cross-engine data executions; thus, this layer integrates big data programming models/engines. Currently, this layer allows the simulation and analysis of the MapReduce model. As maintaining various programming models of big data is essential to support applications that demand different processing mechanisms (MapReduce, stream, etc.), more required components can be added to this layer.

- *Infrastructure*: It contains physical and virtual resources of cloud datacenters. With virtualization mechanisms, hosts in BigDataSDNSim are designed to share their resources among multiple VMs, where each VM has its own memory, storage, and processor characteristics. This layer also maintains network and SDN entities. The fundamental functionalities, deployment, and management of network components and SDN are handled in this layer.

- *CloudSim*: This layer is equipped with the core entities, functionalities, and engine of CloudSim, such as resource provisioning and allocation and event processing mechanisms. It provides essential components to simulate cloud datacenters. The BigDataSDNSim simulator operates on top of this layer, where entities can easily communicate via a discrete-event mechanism.

3.5 Simulation Modeling and Design

This section illustrates the modeling and design of our simulation framework. It first demonstrates its theoretical modeling of BigDataSDNSim. Next, it illustrates multiple proposed designs of BigDataSDNSim, which includes system, policies, and interactions.

3.5.1 Theoretical model

According to the YARN framework [49], the first step required for execution of MapReduce applications is the determination of HDFS data block size. HDFS can be defined as a distributed file storage that breaks down every received data set into small-scale

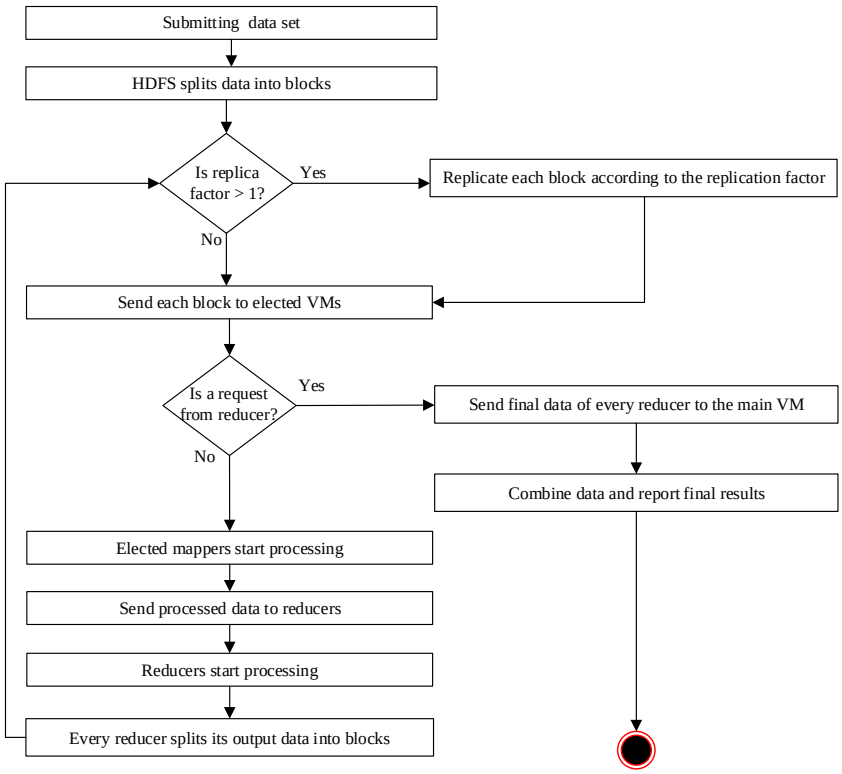


Figure 3.4: BigDataSDNSim MapReduce-HDFS Model

data, referred to as a block, and stores all the blocks on distributed VMs. In a MapReduce YARN-based application, data is broken down into blocks for two times. First, when the HDFS splits a submitted data set into several HDFS blocks to be forwarded to elected VMs, which in turn are assigned to elected mappers for processing. Note that HDFS would only instruct one of the elected mappers to carry out the execution while the others are in a standby mode. The second time is when reducers finish executing where every reducer will split its output into several reducer blocks and distribute the blocks into elected VMs. We denote the former to "HDFS block" and the latter to "reducer block." The two types of blocks use the same block factor bs_c for determining their data block size.

Every generated HDFS block is first copied to elected VMs based on a given replication factor bs_c. One of the elected VMs is selected to run a new mapper for every received HDFS block, while other VMs are in a standby mode. This technique is known as "data locality" where computation is moved to a given data location (e.g., HDFS blocks) instead of vice versa, which helps in reducing MapReduce network loads. Moreover, the output of every reducer is divided into other blocks (reducer blocks) where each block

is replicated and copied to elected VMs based on the same given replication factor bs_c. The HDFS replication mechanism ensures that the processing time of HDFS blocks by mappers is decreased while the reducer replication ensures that the final output of a MapReduce application, which combines data from all reducers, is available at all times. The modeling of BigDataSDNSim MapReduce-HDFS is illustrated in Figure 3.4.

By default, the HDFS engine assigns a default data size bs_d of 128MB to all HDFS and reducer blocks. Every HDFS data block is the input data for every mapper. To tune the number of required mappers, the block size bs_n can be configured. Equation 3.1 is used to select the overall block size where c is a decision variable to denote the demand for changing the default data block size. Let $H = \{1, 2, \ldots, U\}$ be a set of HDFS blocks where each $h \in H$ has a data block size, denoted as $d \in D_h$. Given the size of the MapReduce submitted data set ds, Equation 3.2 is used to compute the total number of HDFS blocks $|H|$. The total number of HDFS blocks $|H|$ is inversely proportional to bs_c, represented as $(|H| \propto (bs_c)^{-1})$. YARN applies division by repeated subtraction to determine the size of every HDFS block d, which means that all elected VMs that store HDFS blocks obtain the same size for all the blocks except for the last one. To properly determine the data size of every block $d \in D_h$, Equation 3.3 is used.

$$bs_c = \begin{cases} bs_d, & \text{if } c = 0 \\ bs_n, & \text{if } c = 1 \end{cases} \tag{3.1}$$

$$|H| = ceil\left(\frac{ds}{bs_c}\right) \tag{3.2}$$

$$Data_h = \begin{cases} bs_c, & \text{if } h < |H| - 1 \\ ds \% bs_c, & otherwise \end{cases} \tag{3.3}$$

Every reducer needs to obtain intermediate data from every mapper. The output size of every mapper cannot be exactly determined because every mapper may produce different patterns of output. For the sake of simplicity, we assume that all of the reducers of a MapReduce application $a \in A$ obtain an equal size input from every

mapper. Let $M = \{1, 2, \ldots, Q\}$ be a set of mappers where each $m \in M$ has a data output, denoted as Out_m. The total number of mappers $|M|$ is equal to the total number of HDFS blocks $|H|$. Let $R = \{1, 2, \ldots, W\}$ be a set of reducers where each $r \in R$ obtains input data, denoted as In_r. Equation 3.4 is used to estimate the data input size In_r for every reducer $r \in R$ by dividing the output size of every mapper $m \in M$ by the total number of reducer $|R|$. For every MapReduce application, there must be at least one reducer; therefore, BigDataSDNSim assigns a single reducer for every requested MapReduce application. However, the number of reducers can be changed if required.

$$In_r = \left(\frac{Out_m}{|R|} \right), \qquad \forall m \in M \tag{3.4}$$

The execution time of every mapper and reducer depends on the number of instructions that is required to be executed on their VMs, which is given in Million Instructions (MI). The execution time also depends on the speed of the central processing unit (CPU) of VMs, which is measured in Million of Instructions Per Second (MIPS). In discrete-event simulators, modeling the speeds, overheads, and sharing factors of CPU, memory, and a hard drive is hard, if not impossible. To correctly capture this type of model, a configurable parameter α is used to capture hidden overheads of VMs when needed. To compute the processing capacity for every VM assigned for any mapper or reducer, Equation 3.5 is used where C_{vm} is the processing capacity of every $vm \in VM = \{1, 2, \ldots, E\}$, $mips(vm)$ is the MIPS speed of vm, and $cpu(vm)$ is the number of CPU cores of vm. Let P_t to be a processing demand of every $m \in M$, G_r be a processing demand of every $r \in R$, and $t \in \{M, R\}$ be a MapReduce task. The execution time of every task t can be computed using Equation 3.6 where $E(t, vm)$ denotes the execution time of t executed in a VM vm.

$$C_{vm} = mips(vm) \times cpu(vm) \times \alpha, \qquad \alpha \in [0, 1] \tag{3.5}$$

$$E(t, vm) = \frac{P_t}{C_{vm}} \tag{3.6}$$

Every mapper and reducer might have different execution times due to CPU sharing mechanisms among all map and/or reduce tasks. In addition, the policy design (e.g., HDFS replica placement, CPU scheduling policies) plays a vital role on the execution time. The feasible option for computing the execution time of a given set of mappers and reducers belonging to a given MapReduce application is to observe the highest execution time of mappers and reducers. To estimate the total execution time $\mathcal{ET}(a)$ of every MapReduce application $a \in A$, Equation 3.7 is used.

$$\mathcal{ET}(a) = \max\{E(t,v)\} + \max\{E(t',v)\}, \quad \forall t \in M, \forall t' \in R, \exists v \in VM \tag{3.7}$$

By default, every MapReduce application $a \in A$ creates three replicas for every block to ensure fault tolerance in case of a VM failure. The replication is performed for every HDFS block and reducer block. Equation 3.8 is used to select the number of required replicas $\Phi \in \mathbb{N}$ where Ω is the requested replication factor and $|VM|$ represents the total available number of VMs. Ω cannot be higher than $|VM|$. Let each $\mathbb{R}_h \in \mathbb{N}$ be the set of replicas for each HDFS block $h \in H$. Given the set of the data of HDFS blocks D_h and the set of HDFS replica size $|\mathbb{R}_h| = \Phi$, every HDFS block $d \in D_h$ is mapped to every corresponding replica $o \in \mathbb{R}_h$ by using a matrix, denoted as $\mathbb{U}_h = D_h \times \mathbb{R}_h$. The data of every replica $u \in \mathbb{U}_h$ will be then transferred from the HDFS to every elected VM.

$$\Phi = \begin{cases} \Omega, & \text{if } \Omega < |VM| \\ |VM|, & otherwise \end{cases} \tag{3.8}$$

Similarly, every reducer divides and replicates its output into a number of reducer blocks based on the overall given block size bs_c. Every block is then replicated according to the replication factor Φ and forwarded to elected VMs. The replication is required to ensure that the final output of every MapReduce application is not lost when some VMs are not available. Let Out_r be a set of reducer output and $B_r = \{1, 2, \ldots, P\}$

be a set of reducer blocks for each $r \in R$ where each $b \in B_r$ has some data, denoted as $z \in D_r$. The total number of reducer blocks $|B_r|$ is inversely proportional to bs_c, represented as $(|B_r| \propto (bs_c)^{-1})$. To determine the total number of blocks $|B_r|$ created by every reducer, Equation 3.9 is used. To estimate the data size of every reducer block $z \in D_r$, Equation 3.10 is used. Let Each $\mathbb{R}_r \in \mathbb{N}$ be the set of replicas for each reducer block $b \in B_r$. Given the set of the data of reducer blocks D_r and the set of reducer replica size $|\mathbb{R}_r| = \Phi$, the data of every reducer block $z \in D_r$ is mapped to every corresponding replica $n \in \mathbb{R}_r$ by using a matrix, denoted as $\mathbb{L}_r = D_r \times \mathbb{R}_r$. The data of every replica $l \in \mathbb{L}_r$ is then transferred from a VM that contains a corresponding reducer $r \in R$ to every other elected VM.

$$|B_r| = ceil\left(\frac{Out_r}{bs_c}\right), \forall r \in R \tag{3.9}$$

$$Data_r = \begin{cases} bs_c, & \text{if } r < |B_r| - 1 \\ Out_r \% bs_c, & otherwise \end{cases} \tag{3.10}$$

A network channel must be established to transfer data from a source VM to a destination VM if they reside in different hosts where each VM may host some MapReduce elements (e.g., HDFS, mappers, reducers). The modeling of channels prevents VMs from overloading any given link existing in its route, which would lead to network congestion. Let $L = \{1, 2, \ldots, K\}$ be a set of links where each $l \in L$ has a bandwidth BW_l and a number of associated channels NC_l passing through. Let $C = \{1, 2, \ldots, Z\}$ be a set of channels traveling through some links where each $c \in C$ obtains a bandwidth BW_c based on the smallest bandwidth of links existing on the route of the channel c. To compute BW_c for every channel c, Equation 3.11 is used where $BW_c(s, d)$ is the bandwidth of channel c from a source VM s to a destination VM d, BW_l is the bandwidth of a link l that a channel c traverses through, and NC_l is the number of channels traveling and sharing a link l.

$$BW_c(s, d) = \frac{min\{BW_l(s, d)\}}{NC_l}, \quad s, d \in VM, \exists l \in L \tag{3.11}$$

For any data to be transferred via an SDN-aware network, it must be encapsulated inside a network flow. We avoid using network packet modeling to prevent our simulation tool from generating millions of network packet objects, which would overload the computing resources of a machine that runs our simulator. Define NF_c as a number of flows that a channel $c \in C$ has and $F = \{1, 2, \ldots, U\}$ as a set of flows where each $f \in F$ has a network bandwidth, denoted as BW_f. As every channel bandwidth BW_c can be shared by flows that travel from a source VM s to a destination VM d, the bandwidth of every flow f can be computed using Equation 3.12.

$$BW_f(s,d) = \frac{BW_c(s,d)}{NF_c(s,d)}, \quad \exists c \in C \tag{3.12}$$

Every MapReduce application requires to transfer data at different times. The transmission of data can be summarized as follows: (1) from HDFS to elected VMs, (2) from mappers to reducers, (3) from reducers to other elected VMs, and (4) from reducers to a VM that reports the final MapReduce results. Let $x \in \{\mathbb{U}_h, Out_m, \mathbb{L}_r, Out_r\}$ be data where each x has an associated flow f to transfer the data of x from one VM to another via the SDN-aware network. To compute the transmission time of every data x, Equation 3.13 is used. The total transmission time $\mathcal{TT}(a)$ of a MapReduce application $a \in A$ is calculated using Equation 3.14. The overall completion time $\mathcal{CT}(a)$ of a MapReduce application a, which is composed of all executions and transmissions, is computed using Equation 3.15.

$$\mathcal{T}(x) = \frac{x}{BW_f(s,d)}, \quad \exists s, d \in VM, \exists f \in F \tag{3.13}$$

$$\mathcal{TT}(a) = \max\{T(x)\} + \max\{T(x')\} + \max\{T(x'')\} + \max\{T(x''')\}\}, \tag{3.14}$$
$$\forall x \in \mathbb{U}_h, \forall x' \in Out_m, \forall x'' \in \mathbb{L}_r, \forall x''' \in Out_r$$

$$\mathcal{CT}(a) = \mathcal{ET}(a) + \mathcal{TT}(a) \tag{3.15}$$

Depending on the value of replication factor, the generated data of every MapReduce application $a \in A$ is different. To compute the total generated data $TD(a)$ of every a, Equation 3.16 is used. The data characteristics generated by different types of MapReduce applications differ from one another. For example, mappers in page-ranking MapReduce-based applications often have a larger size of outputs as compared to the size of their inputs, while word-count MapReduce-based applications follow the opposite [84]. The modeling logic of BigDataSDNSim allows different types of MapReduce applications to be seamlessly simulated by tuning the values of $x \in \{\mathbb{U}_h, Out_m, \mathbb{L}_r, Out_r\}$.

$$TD(a) = \sum_{\forall x \in \mathbb{U}_h} x + \sum_{\forall x' \in Out_m} x' + \sum_{\forall x'' \in \mathbb{L}_r} x'' + \sum_{\forall x''' \in Out_r} x''' \tag{3.16}$$

3.5.2 Implementation

As mentioned earlier, BigDataSDNSim is a discrete-event simulator. Every element that requires communication and cooperation with other elements (e.g., datacenter, resource manager, SDN controllers, etc.) must do so by issuing events that are processed and delivered by the simulation engine. Every event may contain actions to be carried out and data to be used by destinations. The following describes and indicates the property of every component developed in BigDataSDNSim:

- *ResourceManager*: This entity is responsible for the configuration, deployment, and scheduling of worker nodes' resources among competing big data applications. It carries out provisioning mechanisms for new big data applications in a given big data cluster. It tries to reserve the required resources using VM usage statistics. If cluster's resources are insufficient, requests are held in a waiting queue. Once resources become available, the required resources are reserved. The queue is based on a first-come-first-served mechanism. This entity can be instructed with different policies if required.

- *ApplicationMaster*: It is designed to manage the application life-cycle of big data programming models. A new application master is initiated for every new big data application. The initiation of every application master is carried out by a resource manager. The current development of this entity is based on the MapReduce programming model.

- *NodeManager*: This entity is in charge of controlling and monitoring VM resources. Every node manager is coupled with a single VM to track and report the status of its VM. It informs the actual usage of the VM to a resource manager. Such update is used to properly allocate a cluster's resources among big data applications, alleviating resource contention.

- *NetworkNIC*: It is an interface equipped in every node. It is responsible for establishing and maintaining the network connection of a given node. It is similar to the network interface card (NIC) embedded in most of today's devices. The key feature of the interface is to allow the modeling of southbound network management protocols (OpenFlow and Open vSwitch [85] (OVS)) in switches and hosts. Such protocols enable SDN controller(s) to have a full network control of nodes, build routing and forwarding tables, and capture a global network view.

- *SDNController*: This is an entity designed to mimic the behavior of an SDN controller. It provides abstractions for programming and monitoring networks dynamically. By gathering the information from each node (switches and hosts), the controller builds dynamic routing for each host, VM, or application based on a given routing algorithm. The controller is developed to seamlessly shape network traffic for each MapReduce application based on a given traffic policy. The network can be controlled and managed by more than one SDN controllers if required. Network optimization and the reconfiguration of a given network can be easily achieved by implementing smart routing and traffic policies.

- *HDFS*: Every data submitted to a MapReduce application is stored in a distributed manner by the HDFS. The entity divides the data set into several blocks based on given block size. Once data blocks are determined, it copies them to

elected VMs via a network infrastructure. The election of VMs is based on a given HDFS replica placement policy.

- *BigDataTask*: This entity can be shaped in different ways based on a selected big data application. The current design of BigDataTask represents the modeling of mappers and reducers. In the case where a new big data model is required (e.g., stream), the entity can be inherited to obtain the common features that all other big data tasks have in common.

- *Topology*: It is responsible for maintaining a network graph and relations among switches and hosts. An SDN controller uses this entity to obtain network graphs and relations and adjust the network graph if required. Moreover, the SDN controller uses this entity to build routing and forward tables among VMs and applications.

- *Flow*: This is used to facilitate the process of network traffic modeling. A network flow is initiated for every traffic from a source component to a destination component (e.g., hosts, VMs, MapReduce applications). An SDN controller shapes the network traffic of every flow according to given routing and traffic policies.

Figures 3.5 demonstrates an overview of the interactions and workflows among entities throughout the runtime of BigDataSDNSim. The simulator's functionalities are classified into four phases: building required infrastructure, establishing requested MapReduce application(s), carrying out task processing and data transmission, and finally reporting the results of every MapReduce application. The infrastructure is built by parsing a configuration file provided in a JSON format. Once BigDataSDNSim obtains the file, it initiates the corresponding objects of hosts, switches, and network links. At the same time, it establishes the required components, such as the SDN controller and resource manager, and builds a required network topology. Once the resource manager is active, it couples every VM with a node manager for monitoring and reporting purposes.

Shortly after BigDataSDNSim starts, it instructs the resource manager to initiate every requested MapReduce application. The resource manager would create a single

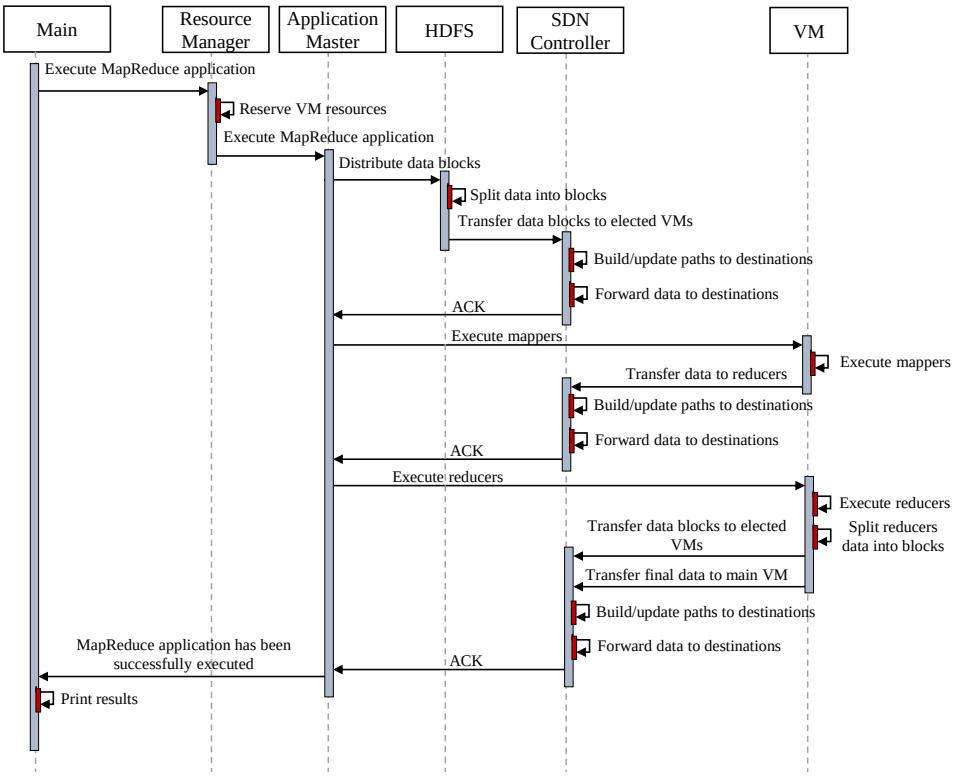


Figure 3.5: Underlying interactions of BigDataSDNSim (overview)

application master for each required MapReduce application. Once every application master is active, every application is fed with the configuration requirements in a CSV file containing different information such as start time, number of mappers, size of flows, and amount of MIs. Each MapReduce application is composed of computational tasks and data flow transmissions, which are carried out in a specific order. The current application logic supported in BigDataSDNSim is based on two processing activities and four transmissions, as depicted earlier in Figure 3.1. For the processing stage to take place, two requirements must be met: (i) the execution logic of map and reduce tasks must be placed into VMs and (ii) mappers and reducers must acquire the whole required data to start execution. The transmission stage, on the other hand, includes four sequence activities (1) dividing initial data into HDFS blocks and transferring the blocks from HDFS to elected VMs; (2) transferring the output of mappers to reducers; (3) dividing the output of reducers into blocks and transferring the blocks to elected VMs; and (4) transferring the output of reducers to a VM, which reports the final MapReduce results.

Any element (e.g., HDFS, mapper, reducer) that requires data transmission must

create a network flow and inject its data to it. Next, it encapsulates the flow inside an event and sends the event to an SDN controller through the implemented discrete-event mechanism. When the SDN controller receives the event, it updates the progress of existing network flows and removes completed ones along with idle network channels. Next, the SDN controller updates or builds forwarding tables based on VM-to-VM communications, with respect to the implemented SDN routing algorithms. If there is no existing route between a source and a destination, the SDN controller will build a route, add a forwarding rule to every node along the route (switches and hosts), create a channel, and encapsulate the received flow inside the channel. In order for the SDN controller to track the completion of flows, it calculates the earliest finish time eft of existing flows, creates an event with an invoking time of eft, and sends the event internally to itself to be intercepted according to the given time of eft. In general, the eft can be computed using Equation 3.17 where $f_r(j)$ is the remaining data of the j_{th} flow to be transferred and $c_{bw}(j)$ is the flow bandwidth of the j_{th} flow.

$$eft = \min\left\{\frac{f_r(j)}{f_{bw}(j)}\right\} \tag{3.17}$$

Once there is no more activity or event to take place, the simulation concludes and results are reported. The report structure reflects the information of MapReduce applications, performance measurement of transmission and processing, and information of SDN forwarding tables. The overall output illustrates every MapReduce application's status, such as submission time, queuing delay, start time, etc. The processing result shows the performance metrics of mappers and reducers (e.g., VM's ID, start time, execution time), while the transmission result demonstrates the statistics of every connection (IDs of source and destination, size of flows, start time, transmission time, etc.). The information of SDN forwarding tables shows the list of traversing nodes for every flow and any changes made to the flow's bandwidth in terms of size and time throughout the transmission period.

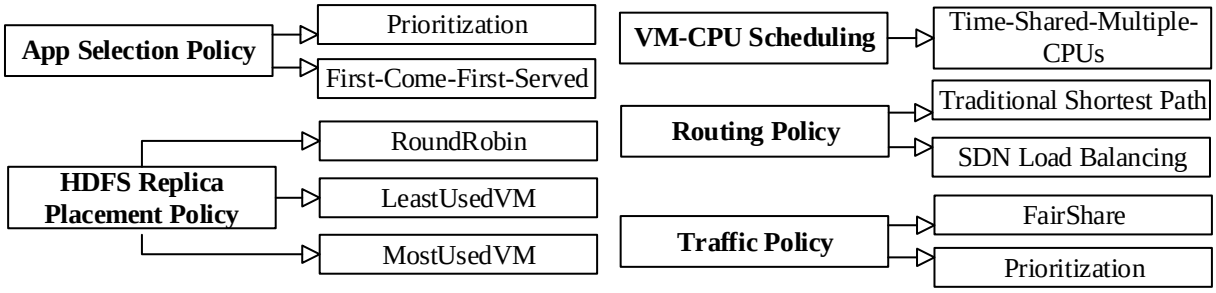


Figure 3.6: Modeling of policies in BigDataSDNSim

3.5.3 Policies design

Developing new policies that dynamically respond to changing behaviors and optimize the performance of different MapReduce applications running in SDN-aware clouds is essential. Therefore, we modeled and developed our simulator to support host-network policies so that different MapReduce policies can be seamlessly implemented. Figure 3.6 shows a sample of different policies currently developed in BigDataSDNSim. The policies are categorized into five groups: app selection, HDFS replica placement, VM-CPU scheduling, routing, and traffic. Several polices are developed for each group. The significance of each group is described as follows:

- *Application selection*: Resources reserved for a big data cluster may be limited. Therefore, an application selection policy is essential to determine the selection criteria based on given QoS (e.g., deadlines). A resource manager queues incoming MapReduce applications and schedules them based on resource availability and given selection policy. There are two different application selection policies implemented within BigDataSDNSim: prioritization and first come, first served.

- *HDFS replica placement*: The replication mechanism is an integral part of MapReduce production systems to ensure reliability and performance. There are many existing policies with different goals to tune the replication mechanism according to a given criteria and constraints (e.g., rack-aware replica placement policy). Therefore, we modeled the HDFS replica placement policy and also included examples of general policies (e.g., round-robin, least used VM, most used VM). New HDFS replica placement policies can be implemented by extending this component.

- *VM-CPU scheduling*: Multiple tasks can be scheduled to be executed in a single VM. The scheduling criteria can be different from one MapReduce application to another. The VM-CPU scheduling was originally implemented in CloudSim. The overall MapReduce processing performance depends on this policy; therefore, we implemented a new VM-CPU scheduling policy called Time-Shared-Multiple-CPUs to enable the use of multiple CPU cores by mappers and reducers.

- *Routing*: In general, the structure of every network is represented as graphs. The network and routing mechanisms of BigDataSDNSim are modeled in a dynamic manner to establish pairwise relationships among different entities (e.g., switches, hosts, VMs, and applications). By such dynamic modeling, BigDataSDNSim enables any network type to be simulated and pairwise relationships and routes to be defined based on different policies (e.g., shortest path). It also enables an SDN controller to manage the entire network routes according to the given criteria. BigDataSDNSim implements the Dijkstra's algorithm [86], which is capable of finding the shortest paths from a single source (e.g., VM, mapper, reducer) to all other destinations based on a single objective, which is a minimum number of traversed nodes. We also enhanced the Dijkstra's algorithm to meet multiple objectives by proposing SDN load balancing algorithm. The objective of our proposed algorithm is to constantly load balance network traffic by finding different paths based on maximum bandwidth.

- *Traffic*: As different types of applications share the resources of a given network, there must be some mechanisms to ensure the network traffic quality for every application. The traffic requirements and flows of every MapReduce application can be different; therefore, the modeling of network traffic policy within BigDataSDNSim is essential to shape the data flows of MapReduce applications according to some QoS criteria. To illustrate the advantage of modeling the traffic policy, we implemented two traffic policies: fair-share and prioritization. New MapReduce SDN-aware traffic policies can be implemented by extending this component.

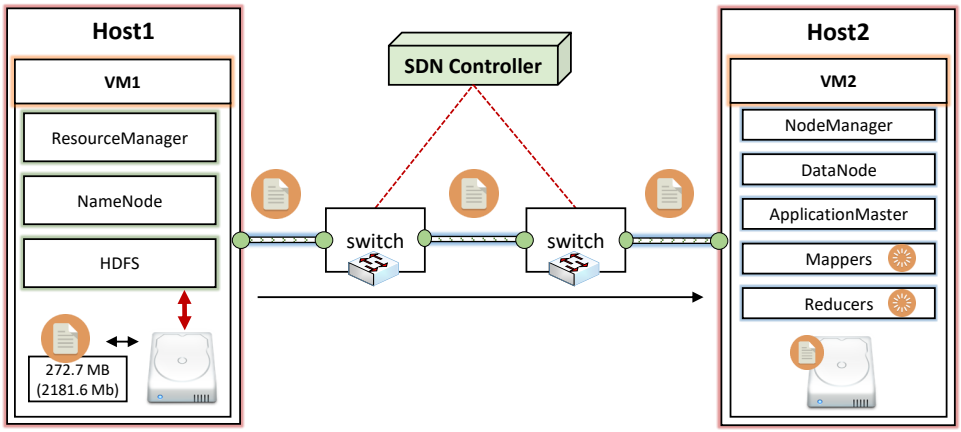


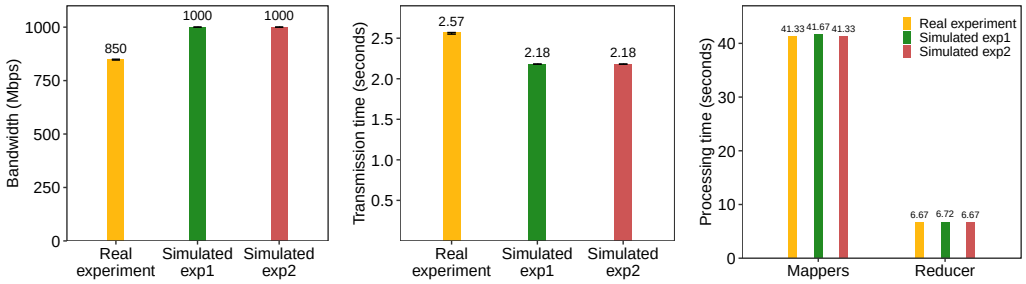
Figure 3.7: Validation setup of the real experiment

3.6 Validation of BigDataSDNSim

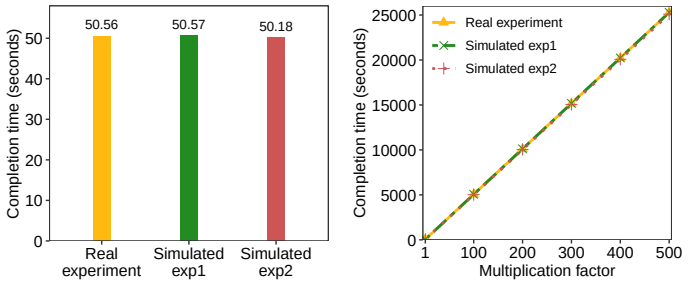
This section reports the simulation accuracy and correctness of BigDataSDNSim by comparing the behavior and results of a real environment that combines MapReduce and SDN with an identical simulated environment that is provided by BigDataSDNSim. The experiment is conducted on two host machines. Each one has Intel Core i77500U 2.70 GHz, 16 GB of RAM memory, and 1000 Mbps of NIC. Each host is equipped with a single guest host (VM) running Linux 4.4.0-31-generic. Each VM has 4 virtual CPUs and 4 GB of memory. Two switches designed by Shenzhen Helor Cloud Computer [16] are used. Each switch has Intel Celeron 1037U (2 Cores, 1.80 GHz), 4GB of memory, and 6 Ethernet ports. The throughput of each port is 1000 Mbps. On each switch, Linux 4.15.0-29-generic and Open vSwitch (OvS) [17] are installed. The OvS is a virtual switch used to allow an SDN controller to instruct and control the switches' data plane via an OpenFlow protocol. An SDN-aware framework called Ryu is used as an SDN controller [87]. The type of cables that connect machines and switches is Ethernet Cat5e. Each cable attains 1000 Mbps of speed.

Table 3.2: Validation configuration

Environment	**Configuration for every VM**				Network Bandwidth
	MIPS	Number of cores	Total number of MIPS	Memory size	
real experiment	3592	4	14368	4GB	850 Mbps
BigDataSDNSim with α overheads	3563	4	14252	4GB	1000 Mbps
BigDataSDNSim	3592	4	14368	4GB	1000 Mbps



(a) Bandwidth comparison (b) Transmission time comparison (c) Processing time comparison



(d) Total completion time comparison (e) Correlation comparison

Figure 3.8: Comparison of the real experiment, simulated exp1, and simulated exp2

For the real experiment, we executed a word count MapReduce application using the Hadoop-YARN framework [49]. We executed the experiment six times and reported the average processing and network transmission times. Moreover, two simulated experiments were carried out: BigDataSDNSim with α overhead (simulated exp1) and BigDataSDNSim without α overhead (simulated exp2). The obtained results of these experiments are compared with the real experiment. The simulated exp1 is intended to slow the simulated VMs to represent performance overheads (e.g., I/O operations, etc.) introduced in the VMs of the real experiment. *The configurations of MIPS, VMs, and network that we obtained from the real testbed was replicated in the simulated experiments.* Also, the same setup scenario of the real experiment (shown in Figure 3.7) is replicated in the simulated experiments. The configuration validation of the real and simulated experiments is illustrated in Table 3.2.

The validation scenario is designed to have one VM that contains HDFS while the

Table 3.3: Configuration for validating MapReduce application

Environment	Total executed MIPS per mapper (mp_{tmi})	Total executed MIPS per reducer (rd_{tmi})	Number of mappers	Number of reducers	File size (HDFS to mappers)	Replication factor
Real experiment	296939	100576	2	1	272.7 MB	1
Simulated experiments	296939	100576	2	1	272.7 MB	1

other VM contains two mappers and a single reducer. The HDFS contains a text file of 272.7 MB that needs to be transferred to the mappers. Once the whole file is transferred, the mappers start processing. In the real experiment, the Hadoop framework tries to place mappers and reducers in the same VM; therefore, there is no need for network transmission between the mappers and reducers. We configured the real experiment to have a single replication of data blocks; thus, the destination VM would only receive unreplicated data blocks. Moreover, the reducer would not transfer its output and data blocks to other VMs since the replication factor is set to one. Data is only transferred once from HDFS to mappers. Table 3.3 shows the MapReduce configuration parameters used in the validation, which is obtained using the Equations 3.18-3.23.

In the real environment, we executed a Linux command on VMs to obtain the number of MIPS nvm, which only indicates the MIPS for a single core. Since the VMs have more than one core, we used Equation 3.18 to estimate the total number of MIPS tnm for every VM where nvm is multiplied by the total number of VM cores vmc. We estimated the real network bandwidth bw of the real experiment by dividing the file size fs by a network transmission time ntt, as shown in Equation 3.19.

$$tnm = nvm \times vmc \tag{3.18}$$

$$bw = \frac{fs}{ntt} \tag{3.19}$$

By using Equation 3.20, the number of MIPS for mappers mp_{mi} is estimated where tnm is divided by the number of mappers mp_n running in the VM. We use Equation 3.21 to obtain the number of MIPS for reducers rd_{mi} where tnm is divided by the number of reducers rd_n running in the VM. Note that reducers only run after all mappers are finished; therefore, the number of MIPS for mappers and reducers are different. Equation 3.22 is used to calculate the total number of MIPS mp_{tmi} executed by mappers where mp_{mi} is multiplied by the maximum execution time mp_{ex} taken by the mappers obtained in the real experiment. Equation 3.23 is used to estimate the total number of MIPS rd_{tmi} executed by reducers where rd_{mi} is multiplied by the

maximum execution time rd_{ex} taken by the reducers.

$$mp_{mi} = \frac{tnm}{mp_n} \tag{3.20}$$

$$rd_{mi} = \frac{tnm}{rd_n} \tag{3.21}$$

$$mp_{tmi} = mp_{mi} \times \max(mp_{ex}) \tag{3.22}$$

$$rd_{tmi} = rd_{mi} \times \max(rd_{ex}) \tag{3.23}$$

Figure 3.8 shows the comparison results of real and simulated experiments. In Figure 3.8a, it can be seen that the bandwidth of the real experiment is 850 Mbps, while the simulated experiments are 1000 Mbps. The small discrepancy is acceptable because there are many factors that hinder the real VMs to achieve the maximum network capacity, such as the speed of CPU, the speed of hard drive (I/O), and the size of memory assigned to the MapReduce application. Figure 3.8b illustrates the transmission times taken to transfer the text file from the source VM (HDFS) to the destination VM that contains the mappers. It can be seen that the real experiment shows a slightly higher transmission time compared with the simulated experiments. Such difference of time can be reduced by changing the value of α parameter in the simulated exp1.

In Figure 3.8c, it can be observed that processing time of mappers and reducer of the real experiment and simulated exp2 are the same. Such similarities are expected since the VMs, mappers, and reducer of both experiments have similar configurations of MIPS. In the same figure, the simulated exp1 has a higher processing time as compared to the real experiment because of the value of α parameter, which makes up the time difference of the network transmission in the real experiment. Figure 3.8d shows the completion time taken in every experiment. By tuning the value of α parameter of the simulated exp1, the simulated exp1 is capable of obtaining approximately a similar completion time as compared to the real experiment. Also, the completion time of the simulated exp2 is closely comparable to the real experiment. Figure 3.8e

illustrates the correlation of the completion time among the experiments. We derive the figure based on the completion time of Figure 3.8d. It can be seen that the completion time has a strong positive correlation, which reveals that the accuracy and correctness of BigDataSDNSim are closely comparable to the real SDN-aware MapReduce environment.

3.7 Use cases of BigDataSDNSim

For the purpose of demonstrating the practicality and advantages of using BigDataSDNSim, we present two use cases: *one replica (1R) and three replicas (3R)*. They shed light on MapReduce performance in terms of illustrating the impact of using HDFS replication mechanisms and the advantages of using SDN. We developed and implemented several MapReduce policies and routing algorithms, which are used in the simulated use-case experiments. Following is the list of policies we modeled and implemented in BigDataSDNSim:

- *MapReduce application selection policy* is configured on a first-come-first-served basis.
- *HDFS replica placement policy* is configured based on round-robin, where each replicated block is forwarded to an elected VM in a sequence-based manner.
- *VM-CPU scheduling* is configured based on time-shared-multiple-CPUs, where every map and reduce task can use multiple CPUs for processing.
- *Traffic policies* are configured as fair-share, where all the network flows obtain an equal amount of bandwidth.
- *Routing algorithms* are configured to use two different routing policies: SDN load balancing and traditional network shortest path. Each routing policy is used in both 1R and 3R experiments and final results are presented and compared.

Both routing algorithms play a vital role in illustrating some advantages of using our simulator. They clearly show the difference between traditional networks and SDN-aware networks in terms of optimizing performance of MapReduce applications. For

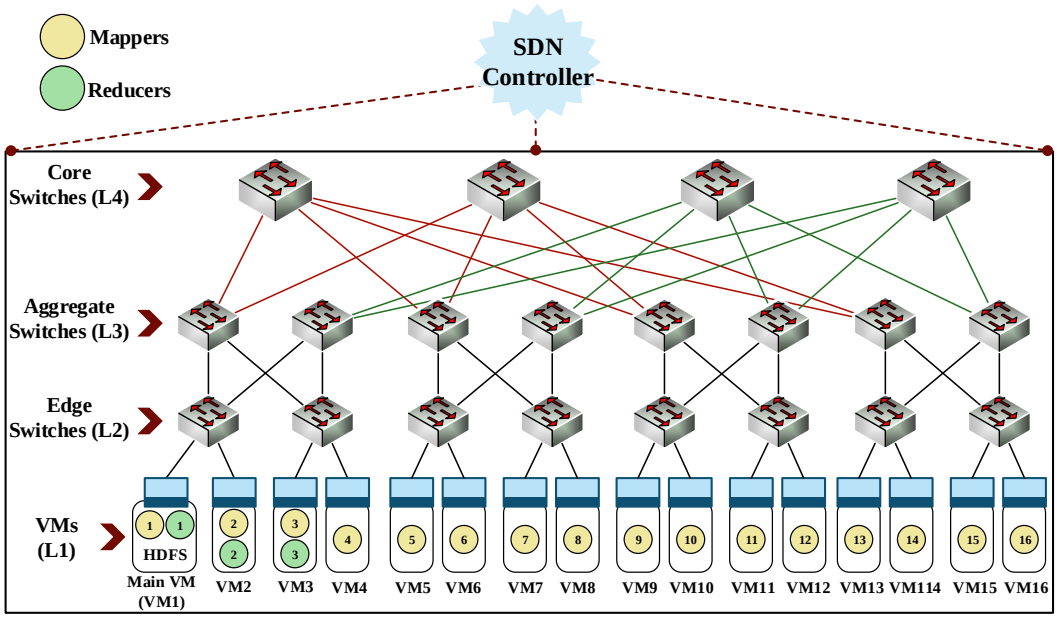


Figure 3.9: Fat-tree topology used in the simulated use-case experiments

Table 3.4: Infrastructure configuration for the use-case

VM	**CPUs**	**RAM (GB)**	**MIPS/CPU**	**Total MIPS**
	4	4	1250	5000
Network	**Link**	**Bandwidth**	-	-
	HDFS to edge switch	3 Gbps	-	-
	VMs to edge switches	1 Gbps	-	-
	edge switches to aggregate switches	1 Gbps	-	-
	aggregate switches to core switches	1 Gbps	-	-

the traditional networks, we implemented the well-known Dijkstra's algorithm [86] to find the shortest paths to destinations based on a minimum number of traversing network nodes. One limitation of traditional networks is that it lacks the selection of different paths from a specific source to a specific destination in a dynamic manner, despite having many elected paths. It finds all elected paths and randomly selects one path where flows of respective source and destination permanently travel via the selected path. On the other hand, SDN-aware networks can program the network on the fly; therefore, we modeled and implemented an SDN load balancing (SDN-LB) algorithm by extending the Dijkstra's algorithm with two objectives of finding routes that obtain a minimum number of traversing node and then finding a route that has the maximum bandwidth among the elected routes. Every time a new flow enters the network, the SDN controller attempts to balance the usage of links by finding an appropriate route for the flow based on our proposed algorithm. For example,

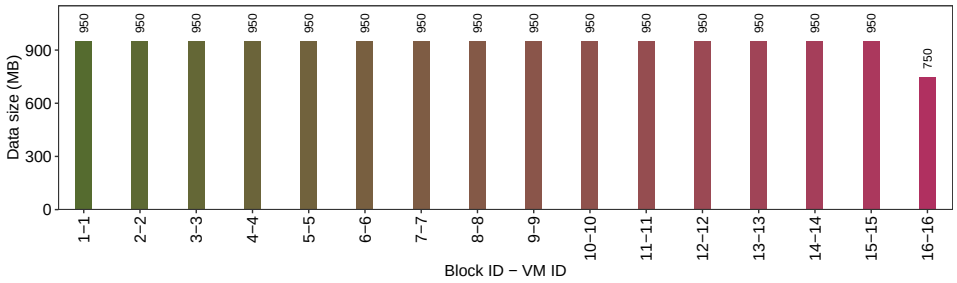
Table 3.5: MapReduce configuration for the use-case experiments

Use-case	Replication factor	Total executed MIPS per mapper	Total executed MIPS per reducer	Number of mappers	Number of reducers	Data size - HDFS to VMs (mappers)	Data size – mappers to reducers	Final data size - reducers to the main VM	Replicated data size - reducers to VMs	HDFS Block Size
Single replica (R1)	1	500000	150000	16	3	$1 \times 15 = 15GB$	12 GB	3 GB	$1 \times 3 = 3GB$	950 MB
Three replicas (R3)	3	500000	150000	16	3	$3 \times 15 = 45GB$	12 GB	3 GB	$3 \times 3 = 9GB$	950 MB

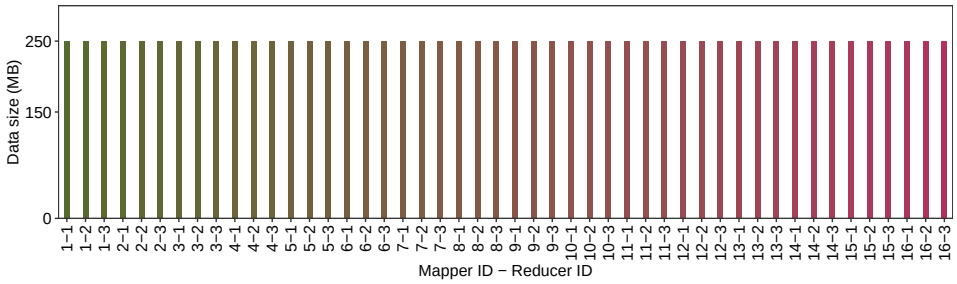
two flows from the same VM can have two different routes to the same destination VM, which would theoretically and practically reduce network transmission time for MapReduce applications.

To set up the simulated environments, we created a single cloud datacenter with the most used fat-tree topology in existing datacenters [88]. Figure 3.9 depicts the physical topology that includes three layers of switches and one leaf layer of hosts containing VMs. There are four core switches (L4), eight aggregation switches (L3), eight edge switches (L2), and 16 VMs (L1) at the leaf of the tree. The network and VMs are configured according to Table 3.4. The structure and link arrangement of the fat-tree topology enables all switches to have additional links to one another so that transmission of data can follow different paths according to a given routing and QoS policies. Every link is configured with a bandwidth of 1 Gbps. The main VM that contains the YARN engine, including the HDFS file system, is connected to the edge switch via a link's bandwidth of 3 Gbps. As the HDFS node is connected to a single link, it would not take advantage of SDN load balancing if its link cannot accommodate a high volume of data. Thus, it is important to have such high bandwidth to demonstrate the impact of SDN load balancing.

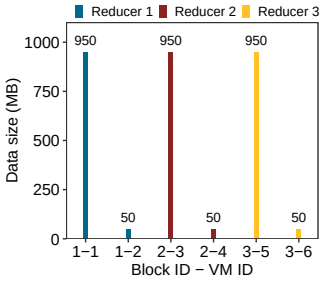
The MapReduce configurations of R1 and R3 experiments are shown in Table 3.5. For R1, the replication factor is set to one, which means that every block of HDFS and reducer is only transferred to a single elected VM. On the other hand, R3 is configured to replicate every HDFS data block to three different VMs along with requesting every reducer to split its final output into blocks and replicate each block according to the replication factor. Such configuration demonstrates the impact of replication mechanisms on the overall MapReduce performance. As the HDFS block size can be altered, we set it up to 950 MB so that the number of mappers is equal to 16. We try to place HDFS, mappers, and reducers in separate VMs as much as possible in order to heavily stress the network within more MapReduce traffic.



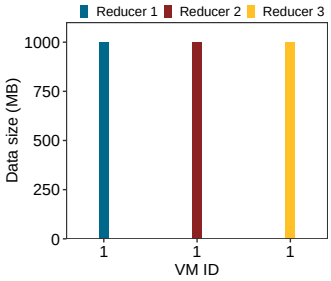
(a) Block size transferred from HDFS to elected VMs (R1)



(b) The output size of every mapper transferred to every reducer (R1)



(c) The output block size of every reducer transferred to elected VMs (R1)

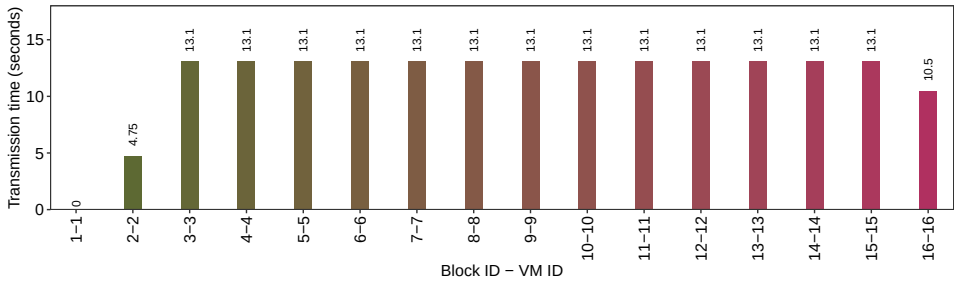


(d) The final output size of every reducer transferred to the main VM (R1)

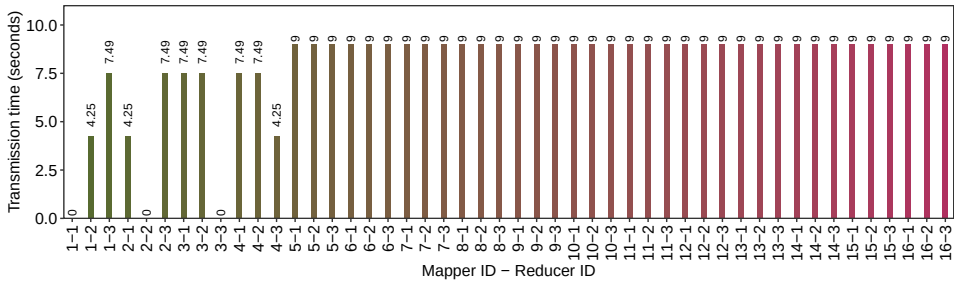
Figure 3.10: Data size of a single replication (R1)

3.7.1 Results of use cases

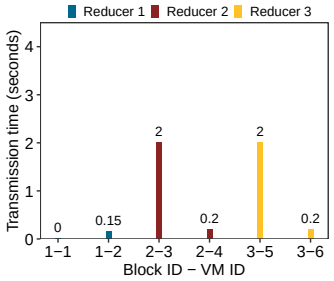
BigDataSDNSim derived the proper data sizes to be transferred from one element to another (e.g., from HDFS to elected VMs) based on Table 3.5 and the MapReduce-HDFS modeling in Figure 3.4. Figure 3.10a demonstrates the size of blocks to be transferred from HDFS, residing in the main VM, to other elected VMs, which are nominated for running mappers. It can be seen that the last block(s) contain fewer data compared to others due to the use of division by repeated subtraction. Figure 3.10b shows the output size to be transferred from every mapper to every reducer. We assume that the output of every mapper is equally divided by the total number of reducers. Figure 3.10c illustrates the block size of every reducer to be transferred to



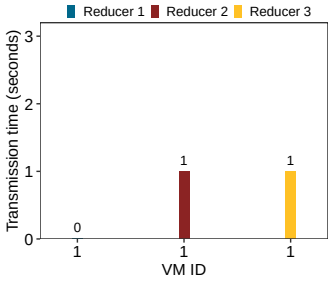
(a) Transmission time of blocks from HDFS to the VMs of mappers (R1)



(b) Transmission time of intermediate data from every mapper to all reducers (R1)



(c) Transmission time of blocks from reducers to elected VMs (R1)

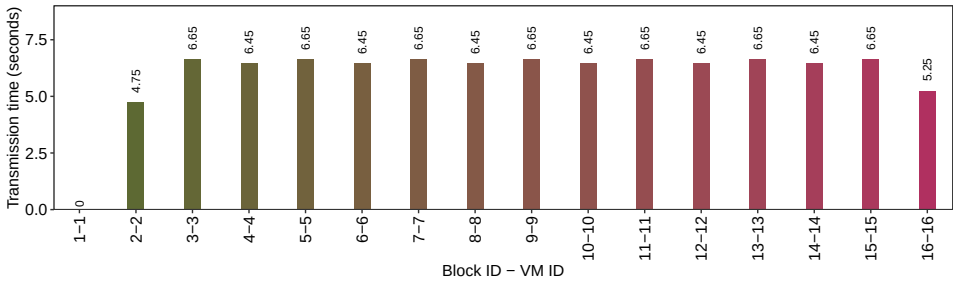


(d) Transmission time of final outputs from reducers to the main VM (R1)

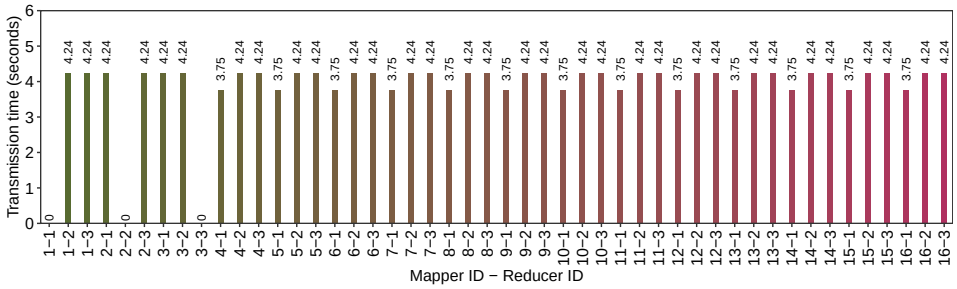
Figure 3.11: Transmission time of MapReduce in a traditional network (R1)

elected VMs. As mentioned earlier, the HDFS requires every reducer to split its output into blocks, replicate every block according to the replication factor, and send every replicated block to nominated VMs. Finally, Figure 3.10d shows the total output size of every reducer to be transferred to the main VM to be combined as the final data.

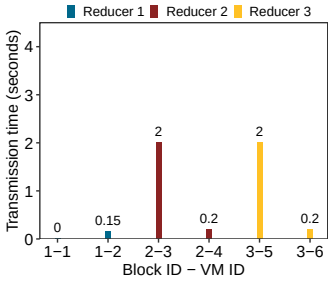
Figure 3.11 and Figure 3.12 illustrate the transmission time of R1 in the traditional network and SDN load balancing, respectively. Some of the transmissions are equal to zero because both the source and destination elements reside in the same VM. Moreover, some of the transmissions are shorter than others. This is because the number of transmissions traveling in the same path is smaller than the number of transmis-



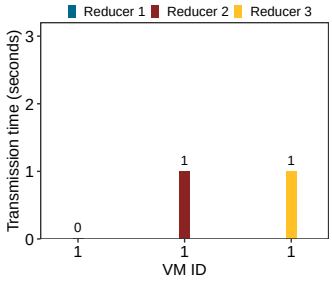
(a) Transmission time of blocks from HDFS to the VMs of mappers (R1)



(b) Transmission time of intermediate data from every mapper to all reducers (R1)

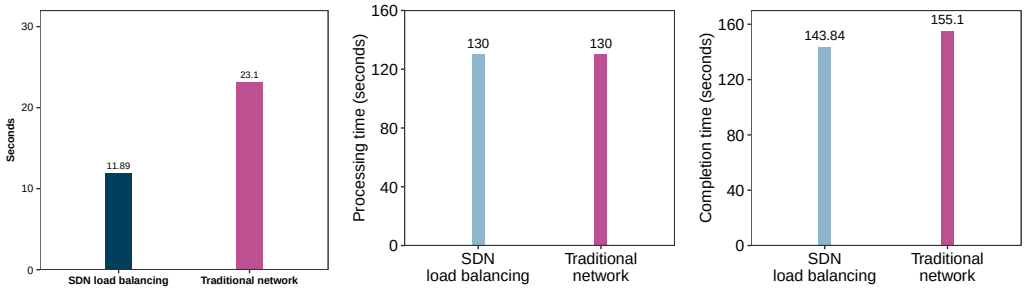


(c) Transmission time of blocks from reducers to elected VMs (R1)



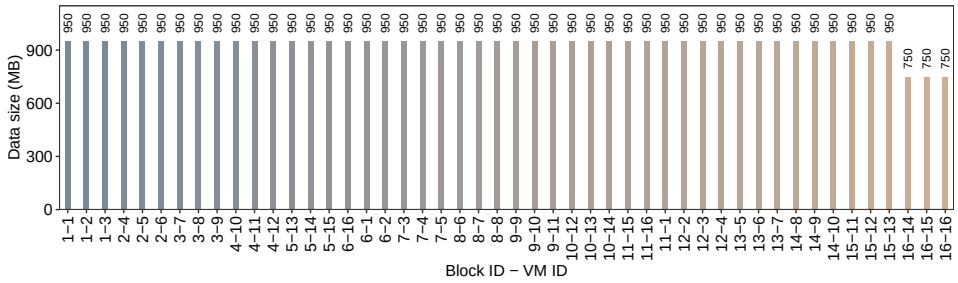
(d) Transmission time of final outputs from reducers to the main VM (R1)

Figure 3.12: Transmission time of MapReduce in an SDN load balancing network (R1)

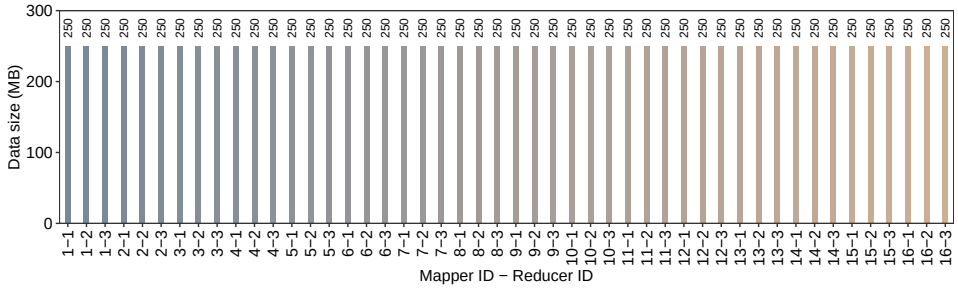


(a) Total transmission time (b) Total processing time (c) Total completion time

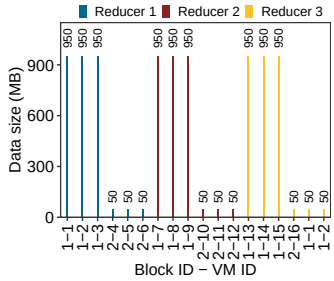
Figure 3.13: Performance comparison of MapReduce in a traditional network and SDN load balancing (R1)



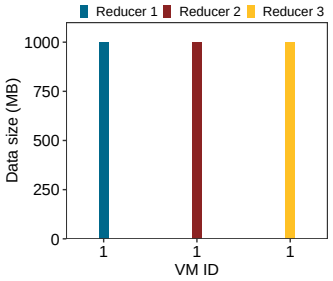
(a) Block size transferred from HDFS to elected VMs (R3)



(b) The output size of every mapper transferred to every reducer (R3)



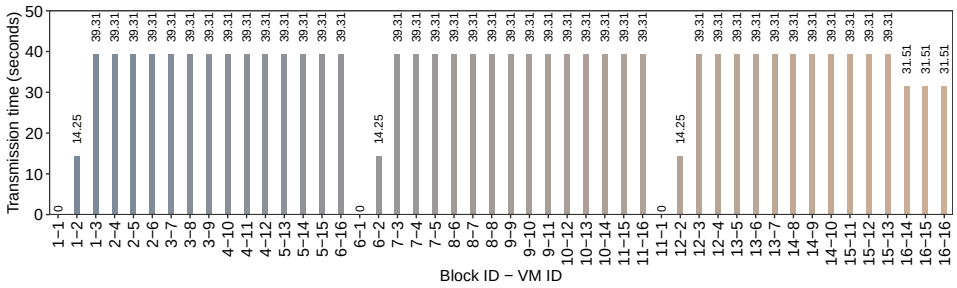
(c) The output block size of every reducer transferred to elected VMs (R3)



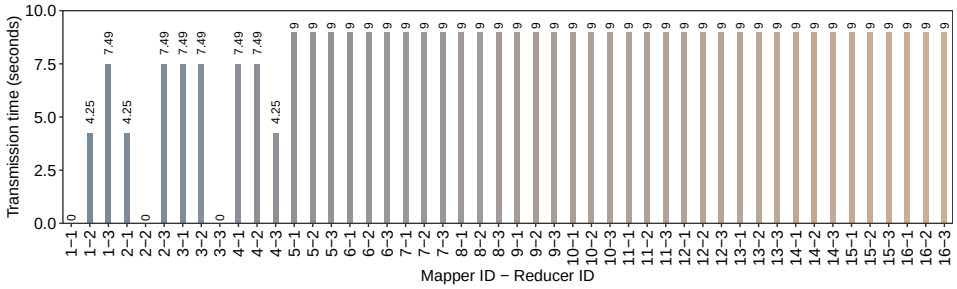
(d) The total output size of every reducer transferred to the main VM (R3)

Figure 3.14: Data size of three replications (R3)

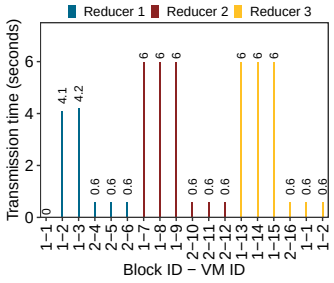
sions traveling via different paths, which results in shorter transmission time. Figure 3.13 shows the performance comparison of the R1 experiment between the traditional network and SDN load balancing. In Figure 3.13a, it is apparent that the SDN load balancing decreases the transmission time by approximately 45% as compared to the traditional network. Figure 3.13b shows the execution time of the mappers and reducers. It is expected similar execution times for the SDN and traditional network as the policy for the CPU scheduling, the distribution of mappers and reducers, and the number of MIPS is the same. The results in Figure 3.13c show the total completion time of R1 in the traditional network and SDN load balancing. The total completion time is determined from the time the HDFS starts transferring data blocks and



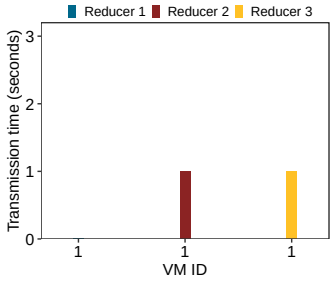
(a) Transmission time of blocks from HDFS to the VMs of mappers (R3)



(b) Transmission time of intermediate data from every mapper to all reducers (R3)



(c) Transmission time of blocks from reducers to elected VMs (R3)

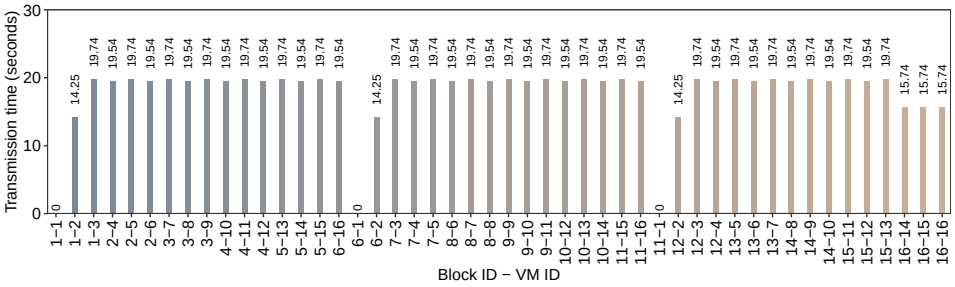


(d) Transmission time of final outputs from reducers to the main VMs (R3)

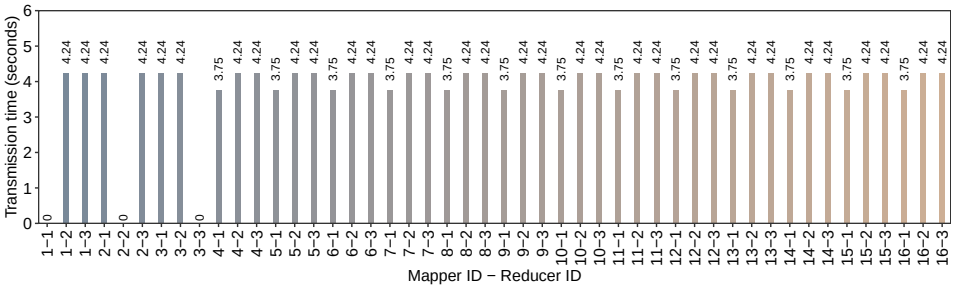
Figure 3.15: Transmission time of MapReduce in a traditional network (R3)

ends when the reducers send their final output to the main VM. It can be seen that the SDN load balancing decreases the total completion time by approximately 8% in comparison to the traditional network.

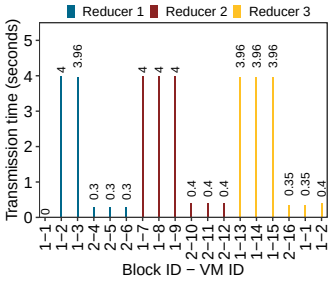
Figure 3.14 illustrates the data size to be transferred from one element to another in R3. The only difference between R3 and R1 is that the former replicates every block of HDFS and reducer three times, which increases the overall data sizes. Figures 3.15 and 3.16 illustrate the transmission time of R3 in the traditional network and SDN load balancing, respectively. Figure 3.17 depicts a performance comparison of R3 between the traditional network and SDN load balancing. It can be noticed that the



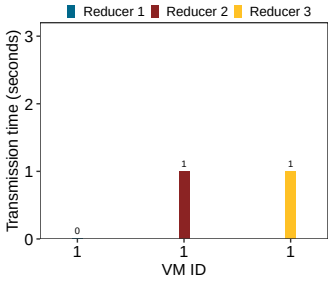
(a) Transmission time of blocks from HDFS to the VMs of mappers (R3)



(b) Transmission time of intermediate data from every mapper to all reducers (R3)



(c) Transmission time of blocks from reducers to elected VMs (R3)

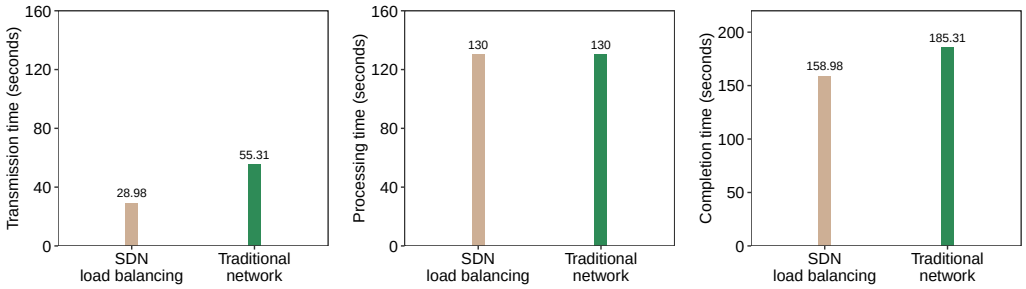


(d) Transmission time of final outputs from reducers to the main VMs (R3)

Figure 3.16: Transmission time of MapReduce in SDN load balancing network (R3)

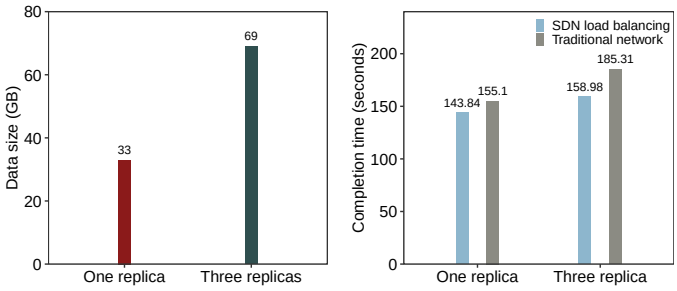
former decreases the network transmission time by approximately 48% as compared to the latter. The SDN load balancing also decreases the total completion time by approximately 14% in comparison to the traditional network.

The comparison of total data size and total completion time between R1 and R3 is shown in Figure 3.18. Figure 3.18a shows that R1 has a smaller data size because its replication factor is set to one. In Figure 3.18b, it can be observed that R1 has a shorter total completion time in both SDN load balancing and traditional network as compared to R3. Moreover, the replication factor is directly proportional to the total completion time.



(a) Total transmission time (R3) (b) Total processing time (R3) (c) Total completion time (R3)

Figure 3.17: Performance comparison of MapReduce between a traditional network and SDN load balancing (R3)



(a) Total data size (b) Total completion time

Figure 3.18: Comparison between R1 and R3

R1 and R3 illustrate an overview of BigDataSDNSim's modeling and features. They stress the significance of the simulator for testing the strengths and weaknesses of new solutions intended for optimizing the performance of MapReduce applications in SDN-aware cloud environments. BigDataSDNSim shows a full picture of the possible states of hypotheses and new proposed solutions. By obtaining the results of every solution, individuals can easily identify and address hidden issues along with ensuring the optimal performance of their approaches and algorithms.

3.8 Conclusions

As the use of simulation-based approaches has been widely applied in numerous fields for analyzing new hypotheses and solutions along with raising awareness of hidden dilemmas, this chapter presents BigDataSDNSim: a novel simulation-based tool that is capable of simulating and evaluating the performance of MapReduce applications in SDN-aware cloud datacenters. The objective of the simulator is to offer holistic

modeling and integration of MapReduce BDMS-based models that are compatible with SDN network functions in cloud infrastructures. BigDataSDNSim provides an infrastructure for researchers to quantify the performance impacts of MapReduce applications in terms of a joint-design of host and network. It contains a variety of application-network policies for diverse purposes (e.g., scheduling and routing), which can be seamlessly extended without a deep understanding of the complex interactions among BigDataSDNSim's components.

In order to demonstrate the correctness and accuracy of the simulator, the performance of BigDataSDNSim is validated with a real MapReduce SDN-aware environment. The validation measures the performance similarities of BigDataSDNSim with the real environment. It reports the comparison results of bandwidth, transmission time, processing time, total completion time, and correlation. Validation results reveal that BigDataSDNSim simulation results are closely comparable to results obtained from real MapReduce SDN-aware environments.

The practicality and advantages of using BigDataSDNSim are demonstrated by presenting two use cases. The use cases focus on MapReduce performance in terms of the impact of using HDFS replication mechanisms and the advantages of using SDN. The performance impacts of SDN load balancing versus traditional networks on MapReduce applications in cloud datacenters are illustrated. The results of the simulated experiments confirm the SDN load balancing decreases the total completion time of MapReduce applications as compared to the traditional networks.

Software availability: The BigDataSDNSim's software with the source code can be downloaded from https://github.com/kalwasel/BigDataSDNSim. A number of examples and tutorials illustrating the use of BigDataSDNSim are given on the web site.

4

IoTSim-SDWAN: A Simulation Framework for Interconnecting Distributed Datacenters over Software-Defined Wide Area Network (SD-WAN)

Contents

Summary

To support the testing and bench-marking of data-driven applications that rely on data ingestion and processing (e.g., Smart Energy Cloud, Content Delivery Networks) across multiple cloud datacenters, this chapter presents a new simulator, named IoTSim-SDWAN. To the best of our knowledge, IoTSim-SDWAN is the first simulator that facilitates the modeling, simulating, and evaluating of new algorithms, policies, and designs in the context of SD-WAN ecosystems and SDN-aware multiple cloud datacenters. This chapter provides an empirical validation by comparing IoTSim-SDWAN with a real-world network environment. Finally, IoTSim-SDWAN is evaluated for network performance and energy to illustrate the difference between classical WAN and SD-WAN environments. The obtained results show that SD-WAN surpasses the classical WAN in terms of accelerating traffic flows and reducing power consumption.

4.1 Introduction

Proliferation of cloud computing has revolutionized hosting and delivery of Internet-based application services. Cloud-based solutions are essential for managing and processing real-time streaming data, such as the data of smart meters connected to millions of households [89, 90]. As streaming data sources are geographically distributed, an overall QoS would vary in terms of data ingestion, transmission, and processing [91]. This variation is dependent upon the location of cloud datacenters in relation to the varied locations of input data streams. The main reason for variable network QoS across data sources and cloud datacenters is the underlying WAN architectures. To obtain optimized network QoS, SD-WAN would be an ideal substitute as its relative SDN solutions have demonstrated significant improvements in several areas such as flow optimization and bandwidth allocation in cloud datacenters [19].

While a number of research has been achieved in proposing and evaluating solutions for SDN located in datacenters, there is a need to address the shortfall in proposing and evaluating new SD-WAN solutions. The benefits are significantly less explored when considering SD-WAN architectures in the context of delivering solutions by networking multiple cloud datacenters. To fill this gap, this chapter presents a new simulation

framework named IoTSim-SDWAN. It models and simulates SD-WAN ecosystems and SDN-aware multi-cloud environments in a discrete-event mechanism. To the best of our knowledge, IoTSim-SDWAN is the first simulator that facilitates the modeling, simulating, and evaluating of new algorithms, policies, and associated design choices in the context of SD-WAN ecosystems and SDN-aware multi-cloud datacenters.

The models, formulas, and framework of IoTSim-SDWAN are empirically validated using real world network data. The validation objective is to illustrate the level of accuracy of bandwidth, network transmission time, TCP/UDP outputs, and overall network delays. The validation is based on three different types of experiments: Iperf3 TCP, Iperf3 UDP, and transferring real data over Ubuntu secure Shell (SSH). The validation results demonstrate that IoTSim-SDWAN is capable of obtaining a high degree of accuracy compared with real networks. Furthermore, this chapter presents two use-case experiments to demonstrate the practicality and capability of IoTSim-SDWAN. The objective of the experiments is to compare the network performance and power consumption of SD-WAN and classical WAN. The obtained results show that SD-WAN surpasses the classical WAN in terms of accelerating traffic flows and reducing power consumption.

The main contribution of IoTSim-SDWAN is the ability to generate different samples of SD-WAN and SDN-aware cloud infrastructures. The simulator can seamlessly evaluate and report the impact of new solutions. The accuracy and correctness of IoTSim-SDWAN are validated using the simulator's output results with the results obtained from an equivalent, real SDN-aware environment. The validation results demonstrate that IoTSim-SDWAN is closely comparable with real environments. This chapter also demonstrates the practicality and advantages of IoTSim-SDWAN by presenting a use-case evaluation experiment, which considers the comparison of performance and energy-consumption of SD-WAN and SDN-aware cloud datacenters as compared to classical WAN and cloud datacenters respectively. In summary, the main contributions of this chapter are as follows:

- Proposing a novel framework that simulates and models the SD-WAN and SDN-aware datacenters

- Accurate modeling of TCP and UDP protocols in addition to network delays in IoTSim-SDWAN
- Proposing an SD-WAN routing technique to dynamically compute the best route for every network flow together with proposing a coordination scheme for SD-WAN and SDN controllers
- Empirically validating IoTSim-SDWAN with a real-world network environment

The rest of this chapter is divided into several sections as follows. Section 4.2 discusses design criteria and motivation. The design and modeling capabilities of IoTSim-SDWAN is presented in Section 4.4. Section 4.3 illustrates the most rel event related work. Section 4.5 describes the empirical validation and accuracy of IoTSim-SDWAN. Experiments are presented in Section 4.6 to show how IoTSim-SDWAN can contribute to multi-CDC design issues for smart applications. Section 4.7 concludes the chapter and highlights our future plans.

4.2 Overview

To better understand the difficulty of creating, evaluating and banchmarking performance using a bespoke testbed for data-driven multi-cloud applications, consider the example of a smart energy cloud. It is well understood and documented that energy companies collect significant amounts of data, possibly beyond the amount that they can manage and analyse. The problem of big data arises for these companies due to large scale deployment of smart meters and smart grid devices (e.g., transformer sensors, circuit breaker sensors, voltage regulator sensors, and other assets that have the ability to communicate their status back to the private/public CDC-based control center in real-time). Considering the sheer number of configurable elements within this scenario, there is no real practical solution to deriving optimal parameter values apart from modeling through industry recognised benchmarks. Even if such a model yields non-optimal results, sufficient information will provision the engineer with a clearer understanding of network QoS requirements to achieve appropriate Demand Response (DR) latency in the presence of data-streams exhibiting volatile behaviours.

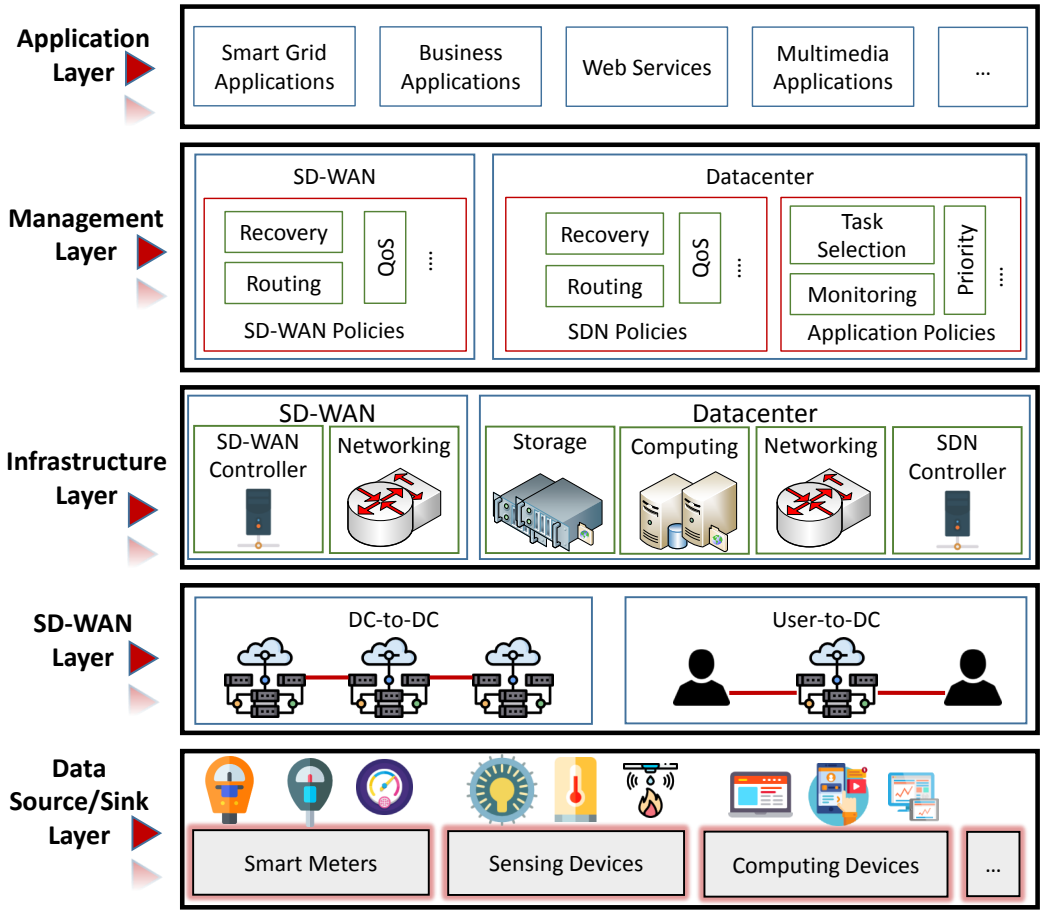


Figure 4.1: Architecture of distributed enterprise ecosystems and SD-WAN

The (classical) WAN is a core communication layer where it provides the fundamental building block for enabling secure and salable shared resource access across geographically dispersed distributed systems. However, the main drawback of a WAN is that it typically exhibits under resource utilization (30-40% [16]). Losing 60% of network utilization due to the static nature of WAN network management (inability to manage utilisation in varying traffic flows) is not acceptable for modern, resource aware, smart digital infrastructures. To improve today's WANs, new software based approaches (SD-WAN) are adopted by commercial and state organisations. The earliest integration of an SD-WAN ecosystem for improved network utilization was by Microsoft [17] in 2013 followed by Google in 2014 [16]. Both Microsoft and Google leverage SD-WAN solutions to accelerate the process of copying large amount of data across, and between, datacenters while improving network performance, coordination, traffic engineering and overall resource optimizations.

A simplified view of the layered architecture of distributed enterprise ecosystems that utilise SD-WAN is highlighted in Figure 4.1. Applications can directly leverage SD-WAN capabilities to better deliver services. For example, multimedia providers can use an SD-WAN to efficiently interconnect their distributed datacenters to enable optimized delivering of geographic aware streamed videos to customers. This provides improved customer satisfaction (e.g., less transmission time), reduces operational costs (e.g., optimum average network utilization), and efficiently manages the entire network infrastructure (e.g., end-to-end view of per-flow performance).

SD-WAN can be deployed in a variety of ways. For example, an enterprise can leverage SD-WAN across global private networks (e.g., Aryaka [92], Silver-Peak [93]). Such enterprises may improve user experience, such as prioritizing network traffic based on user's demands. Another SD-WAN deployment option is to allow global brands (e.g., Google [16]) to manage their own resources in the context of mass data migration and storage requirements. The relation between enterprise ecosystems and SD-WAN, as depicted in Figure 4.1, is demonstrated as the following layers:

- *Data source/sink* maintains various devices that generate and receive data. The devices include, but are not limited to, IoT devices (e.g., smart meters, sensing devices), and computing devices (laptops, Raspberry Pi). These devices can transfer data to the respective datacenters while the datacenters can instruct and send data to devices. For devices to access SD-WAN, they must be connected to their nearest network gateways.

- *SD-WAN* provides a two-way approach of deployment and communication. The first enables the communication of datacenter-to-datacenter (DC-to-DC) and assumes responsibility for exchanging large amounts of data across geographically dispersed datacenters. Secondly, it enables the communication between users and distributed datacenters (user-to-DC). Both types require different management and policies for defining and modifying SD-WAN networks in real-time based on changing traffic flow requirements. The current stage of our work presented in this chapter (IoTSim-SDWAN) focuses on the modeling of DC-to-DC.

- *Infrastructure* consists of datacenter and SD-WAN hardware supporting require-

ments. Every datacenter contains storage, computing, and networking equipment in addition to the deployment of SDN controller(s) for managing the internal network. SD-WAN requires networking equipment and SD-WAN controller(s) for managing the network between distributed datacenters in addition to end users.

- *Management* provides the ability to control, configure, and program the resources of datacenters and SD-WAN. This enables every datacenter to enforce different policies on applications, such as priority and task selection. This also allows every datacenter to program and monitor its internal network through the use of SDN. The management layer also offers features to SD-WAN for controlling, programming, and reshaping SD-WAN network traffic.

- *Application* facilitates the interactions with different type of services and applications. Every enterprise requires software enabled services to reinforce efficiency and productivity. This layer provides an abstraction from the underlying layers, allowing enterprises to focus on the development and deployment of efficient applications and services while maintaining minimal knowledge of the underlying layers.

IoTSim-SDWAN is based on the design criteria of the above layers. The current stage of IoTSim-SDWAN satisfies the requirement, design and implantation of the three layers: SD-WAN, infrastructure, and management. IoTSim-SDWAN also supports a generic design for the application layer where the tools support users in designing and implementing the relations and workflows of their applications. For example, implementing the behaviours and interactions of web applications according to a given web architecture (e.g., middleware systems, databases). The modeling approach of IoTSim-SDWAN is generic and flexible where any type of datacenter and SD-WAN topology can be simulated. IoTSim-SDWAN also supports the modeling and evaluation of network performance and energy consumption.

4.3 Related Work

For several decades, there have been numerous solutions for tackling a traditional WAN's issues in performance management (e.g., slow packet delivery, waste of network resources, routing complexity) together with monitoring and improving network performance and QoS. For example, the emergence of Multiprotocol Label Switching (MPLS) [94] in the early 2000s aimed at increasing network bandwidth and improving network packet delivery overcame the shortcomings of classical WAN traffic engineering along with improving QoS for latency-sensitive applications. Although MPLS has become the de-facto standard for WAN traffic engineering since its discovery, it still encounters major obstacles such as long setup times, inflexible in the presence of dynamic changes in network conditions, and a lack of dynamic routing mechanisms. The characteristics of modern applications coupled with rapid evolution of large systems, the classical WAN fails to cope. This is evident when considering today's application requirements, such as application-aware traffic engineering, obtaining real-time network changes, on-the-fly network re-configuration/provisioning, and real-time bandwidth reservation. Edge based streaming services and on-demand high volume data migration services all place these types of requirements on existing distributed systems.

There are many simulation tools have been developed to aid researchers and developers to evaluate new algorithms for the management of different computing resources and systems in a controllable and repeatable manner. These tools can be categorized into four main groups relevant to IoTSim-SDWAN's work: (i) *Cloud Simulators* that model behaviour of cloud components such as datacenters and virtual machines along with scheduling and provisioning policies; (ii) *Network Simulators* which focus on the modeling and simulating of network systems in different computing environments; and (iii) *Cloud-based Application Simulators* that capture and simulate the behaviours, workflows, and dependencies of various applications, such as MapReduce and web applications; and (iv) *Edge Simulators* that simulate the characteristics and behaviours of edge environments (e.g., IoT devices, edge devices, computing and analytic operations).

CloudSim [1] is a discrete-event simulation tool that enables the modeling and simu-

lation of cloud-related systems. It supports the modeling of cloud system components such as datacenters and virtual machines (VMs) in addition to resource provisioning policies with the aim of optimizing the performance of cloud infrastructures. GreenCloud [70] is a simulator designed for energy-aware cloud datacenters. It evaluates the energy consumption of datacenter components such as servers. iCanCloud [71] is a cloud-based simulator designed to conduct large-scale experiments to ease the process of integrating new policies for cloud brokers. NetworkCloudSim [68] is an extension of CloudSim with the aim of providing network modeling and also provides a generalized application model to allow the evaluation of scheduling and resource provisioning policies. These tools provide cloud infrastructures and some of network capabilities but fail to model SDN and SD-WAN infrastructures.

CloudSimSDN [2] is a simulation framework for SDN-aware cloud environments developed on top of CloudSim. Its objective is to model SDN environments as well as reduce the energy consumption of hosts and network components by evaluating scenarios with different management policies. However, it is not flexible in terms of simulating different network topologies and does not adapt to varying application traffic policies. Moreover, it lacks the modeling and simulation of SD-WAN environments.

BigDataSDNSim [95] is a simulator developed on top of CloudSim and CloudSimSDN. BigDataSDNSim models and simulates big data analytics applications of MapReduce by allowing reducers and mappers that are spread across multiple hosts to communicate with one another over SDN-aware cloud datacenters. This approach also allows the simulating of any type of network topology along with providing greater flexibility for implementing new MapReduce SDN-aware scheduling techniques. Nevertheless, it is limited within a single datacenter and lacks SD-WAN ecosystem properties to interconnect distributed datacenters.

Mininet [96] is a lightweight network emulator that uses OS-level virtualisation for prototyping large networks with the resources of a single laptop. It supports the emulation of SDN and networked systems. It can be used to evaluate the performance of SDN. Similarly, NS-3 [97] consists of a discrete-event network simulator that allows the emulation of real world protocols in both IP and non-IP based networks, but it lacks the modeling of SDN environments. These tools do not allow the modeling

Table 4.1: Comparison of related simulators

Simulators	Features								
	Cloud support	Traditional n/w support	Multi-data-center comm.	SDN support	SD-WAN support	TCP/UDP n/w protocols	Dynamic n/w adaptability	Heterogeneous n/w topology	Power modeling
CloudSim[1]	✓								
GreenCloud[70]	✓								✓
iCanCloud[71]	✓								
NetworkCloudSim[68]	✓	✓							
CloudSimSDN[2]	✓	✓		✓					✓
BigDataSDNSim[95]	✓	✓		✓			✓	✓	✓
Mininet[96]		✓		✓		✓	✓	✓	
NS-3[97]		✓				✓	✓	✓	
IoTSim-SDWAN (Proposed)	✓	✓	✓	✓	✓	✓	✓	✓	✓

and evaluation of cloud environments and features, such as virtual machines allocation policies and application workload distribution. They also lack the support of SD-WAN environments.

EdgeCloudSim [98] and IoTSim-Edge [99] are simulators designed to imitate the environments of edge computing and IoT. EdgeCloudSim focuses on the modeling of some behaviours of edge computing and IoT devices, such as network communication, mobility, and processing operations of edge devices. IoTSim-Edge models many behaviours and mechanisms of edge and IoT devices, such as network and edge protocols, heterogeneity, mobility, and power consumption. However, these tools lack the modeling and simulation of SDN and SD-WAN environments.

The summary of aforementioned simulators are provided in Table 4.1. To the best of our knowledge, there is no existing tool capable of simulating workloads on cloud environments that span several datacenters, each exhibiting a specific network topology enabled through SDN. IoTSim-SDWAN is a novel simulator that allows the modeling of cloud-specific application executing across heterogeneous datacenters with SDN-aware support both within the datacenters (local) and between them (WAN).

4.4 Design of IoTSim-SDWAN

This section describes the model and framework designs of IoTSim-SDWAN. Sub-section 4.4.1 illustrates various factors that affect network performance, for example, the impact of TCP/UDP protocols and delays produced by different approaches. We mathematically model the behaviour, relationships, and variables of SD-WAN in sub-section 4.4.2. Sub-section 4.4.3 demonstrates the network modeling of SD-WAN and

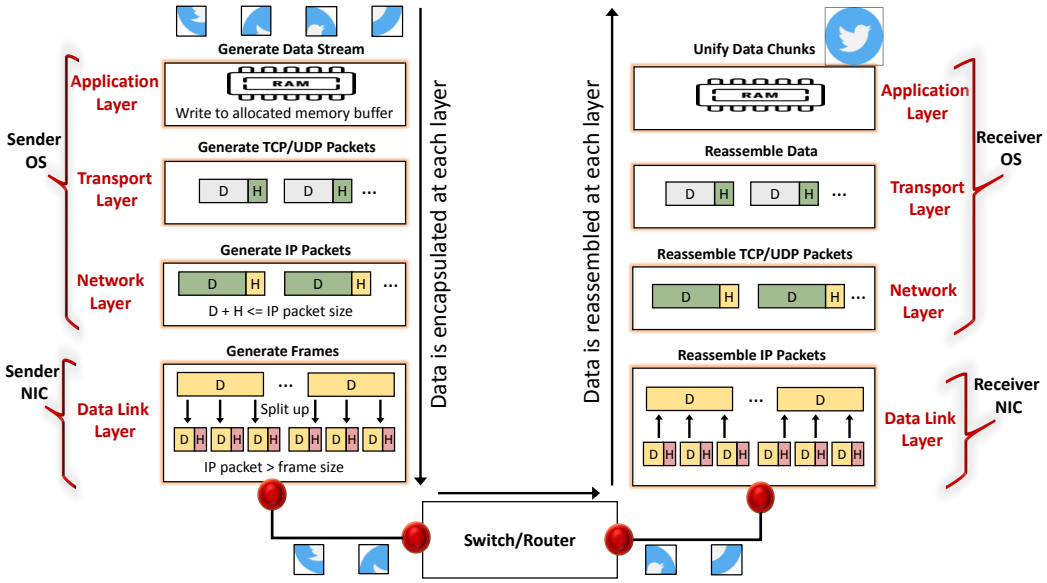


Figure 4.2: Procedure of data transmission based on the TCP/IP model

classical WAN based on graph theory along with presenting our proposed Shortest Path Maximum Bandwidth (SPMB) routing algorithm and our proposed coordination scheme for SD-WAN and SDN controllers. Sub-section 4.4.4 illustrates the system structure overview and physical properties of IoTSim-SDWAN and demonstrates the interactions among the components of IoTSim-SDWAN in a simplified form.

4.4.1 Considerations for performance modeling

End-to-end network performance between endpoints depend on many internal structures of applications, systems, and networks. Even with two directly connected nodes, theoretical network measurements may not achieve the same values as practical network measurements. The dynamic changing status and the underlying capabilities of such structures are hard to determine. However, we attempt to list and illustrate the factors that play important roles in network performance obtained from our real network observations. Such factors are captured and modelled in IoTSim-SDWAN.

In figure 4.2, a conceptual network model (known as a TCP/IP model [100]) is presented to illustrate how data is being transferred from a sender to a receiver. Each layer can affect the performance of data transmission in different ways. Each layer appends a header or footer with the passing data for identification, flow control and error control purposes. Based on the underlying protocols, every layer ensures that

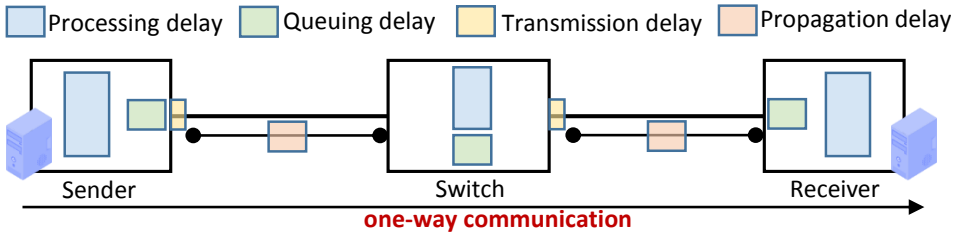


Figure 4.3: Overview of network delays

the data size does not exceed its total allowed size; otherwise, the data is split into a number of smaller packets/frames with each packet/frame tagged with a new header. Such techniques increases the data size which results in longer transmission time.

Figure 4.3 shows the total delay introduced when transmitting data in a simplex communication mode (one-way). The main delays can be characterized into processing, queuing, transmission, and propagation. The processing delay differs from one node to another. For example, a sender's CPU needs to acquire data from a hard drive (first delay), apply some operations on the data (second delay), and store the data in memory (third delay). A traditional switch has different processing delays where it read the headers of incoming packets and finds an output port/link, subject to the time taken for searching and finding a record in its database table that matches the header's details. In case of SD-WAN and SDN-aware mode, most of network processing delays are delegated to an SD-WAN and SDN controller.

The queuing delay is the waiting time of each packet before it is put on a link or processed by a given node. There are many factors that affect the waiting time, such as available network bandwidth and a node's processing capabilities. The transmission delay is the time taken to place the whole data of a given packet onto a link. If the packet size originated from the network layer is larger than a frame maximum size (maximum transmission unit - MTU) size, the data link layer will split-up/fragment the packet into multiple frames (described in sub-section 4.4.2). Another way to describe the transmission delay is the total time taken to push all consecutive bits belonging to a frames of single packet to a given link. The propagation delay is the time taken to deliver bits of frames belonging to a given packet between two adjacent nodes (e.g., sender and switch in figure 4.3). The propagation delay is subject to the physical length and propagation speed of a given cable.

Network speed is not the only factor that affects the transmission time of given data. There are a number of contributing factors that are involved in determining the transmission time, such as the speed of network interface controllers (NICs), allocated memory sizes of senders and receivers, and the hard drive writing and reading speeds of senders and receivers. Moreover, applications cannot use a given network separately, they must somehow share the network according to a given set of policies (e.g., fair-share, prioritizing). Such sharing techniques affect the transmission time. Therefore, considering such performance requirements in IoTSim-SDWAN is important to achieve a realistic network performance as much as is feasible.

4.4.2 Theoretical model

The transport layer of TCP/IP model (as shown in figure 4.2) plays a critical role in today's networks (e.g., SD-WAN, WAN, datacenter networks, LANs). This layer contains two common protocols: UDP and TCP. UDP is critical for many today's applications (e.g., multimedia) where minimum rate of transmission time is more important than sending data reliably. On the other hand, TCP is important for many applications (e.g., e-commerce), which require improved reliability for transmitting data while tolerating a degree of delay. In essence, TCP provides a degree of reliability at the expense of delay while UDP dispenses with any reliability effort to improve transmission delays which may result in lost messages (increased error). In this way, UDP is considered to be faster than TCP protocol as it does not enforce any overhead mechanism found in TCP (requesting if packets have arrived and re-sending of lost packets). TCP is often seen as a streamed interaction using sliding window protocols to improve reliability whereas UDP is seen as a one-off send without sender considering any reliability issues.

The packet payload size at the transport layer is subject to the system and application performance of senders and receivers. The payload size of every TCP packet is difficult, if not possible, to accurately quantify on the fly due to consistently changing factors (e.g., the write speed and allocated size of sender's memory, the read speed and allocated size of receiver's memory). The payload size is also subject to an advertised TCP sliding window size originated from the receiver at some point during a given

network communication. There are some techniques that can be used to increase the window size (a.k.a TCP window scaling [101]) where receivers can handle more data than the traditional window size can. A UDP payload size is not restricted where a respective receiver consistently receives packets with no restrictions on memory size. If too many UDP packets arrive at a receiver than the receiver can handle they are simply dropped and forgotten. For sake of simplicity, we used the concept of averaging to give an approximation of TCP and UDP payload size, this is left for users to decide according to their specific application dependent scenarios.

Table 4.2 illustrate the symbol used in modeling IoTSim-SDWAN. Given the data size $ds(i)$ of ith data and an average packet payload size pl_s at the transport layer, the total number of packets pn for the given data is calculated using Equation 4.1. When TCP/UDP packets are handed to the lower network layer, they would be encapsulated into IP packets where the IP addresses of source and destination are stamped. If TCP/UDP packets at the transport layer are larger than the predefined size of IP packet at the network layer, they would be broke up into multiple IP packets. For simplicity, we assume that all packets at the transport layer are less than or equal to the IP packet size at the network layer.

$$pn = \frac{ds(i)}{pl_s} \tag{4.1}$$

Each packet at both the transport and network layers must be tagged with a unique identification header. Equation 4.2 is used to obtain the header size $hs(p)$ of every

Table 4.2: Modeling Notation

Symbol	Description	Symbol	Description
H	Set of all hosts	$c(i,j)$	A channel from sender i to receiver j
C	Set of all channels	$tc(n)$	Total number of channels carried out by nth link
F	Set of all flows	$c_{bw}(i,j)$	Channel bandwidth between sender i and receiver j
N	Set of all networks (datacenters & SD-WAN)	$l_{bw}(n)$	Link bandwidth for nth link
pn	Total number of packets	f_n	Number of flows between endpoints
$ds(i)$	Data size of ith data	$f(si,dj)$	Number of flows between sth application on ith node and dth application on jth node
pl_s	Average packet payload size	f_{bw}	Flow bandwidth
$hs(p)$	Header size of every packet	f_s	Total data size of a flow
tp_h	Transport header size	d	Total network delay
ip_h	Network (IP) header size	d_p, d_q, d_t, d_p	Processing, queuing, transmission, and propagation delays respectively
$ts(p)$	Total size of each packet	$t_d(f)$	Total delays of given flow
nf	Total number of frames	$tr(f)$	End-to-end transmission time of a flow
mtu_s	Average MTU size	tn	Routing of traditional networks
tf	Size of each frame	$r(n_i,n_j)$	Link connecting node i to node j
$dl_{ht}(p)$	Header and footer sizes of the data link layer	nc	Total number of channels

packet p where tp_h is the transport header size and ip_h is the network (IP) header size. Equation 4.3 is used to determine the total size *ts(p)* of each packet p that includes the average packet payload size pl_s and header size $hs(p)$

$$hs(p) = tp_h + ip_h \tag{4.2}$$

$$ts(p) = pl_s + hs(p), \forall p \in \{1, 2, ..., pn\} \tag{4.3}$$

When the network layer passes IP packets to the lower data link layer, the packet will be encapsulated into a frame. If the IP packet size is larger than the MTU, the packet will be segmented into multiple frames according to the maximum MTU size [102]. The MTU is not always constant due to the nature of networks. To simplify the MTU size, an average MTU size mtu_s is used. Equation 4.4 is used to calculate the total number of frames nf that packets can be fragmented into. The size of each frame tf is computed using Equation 4.5 where $dl_{ht}(p)$ is header and footer sizes appended by the data link layer.

$$nf = \frac{\sum_{n=1}^{pn} ts(p_n)}{mtu_s} \tag{4.4}$$

$$tf = mtu_s + dl_{ht}(p) \tag{4.5}$$

In order to send the data from a sender i to a receiver j, a channel *c(i,j)* must be established which traverses throughout all the underlying nodes of a selected path. Using the concept of channel makes the network bandwidth management easier where all hosts H can share the network based on a given traffic policy (e.g., fair share, prioritizing). The total number of channels nc in a given SD-WAN and SDN-DC network is determined using Equation 4.6.

$$nc = \sum_{i,j \in H} c(i, j) \tag{4.6}$$

Every channel must pass through certain links that connect senders and receivers.

Every link has its own available bandwidth that constantly changes according to the number of shared channels. The initial bandwidth size of a given link l is determined by taking the minimum bandwidth value of its two directly connected nodes' network interface cards (NICs). The bandwidth for each NIC is obtained from a given topology file in a JSON format (refer to figure 4.8). For every node that has more than one connected link, it must attach a separate NIC for each link. Every link can have different numbers of channels. Therefore, Equation 4.7 is used to compute the total number of channels *tc(n)* carried out by a given nth link.

$$tc(n) = \sum_{c \in C} l(c) \tag{4.7}$$

A network can be congested when the transmission for a set of packets belonging to a given application/data is not restricted. To avoid network congestion, we assign bandwidth to every channel by taking the minimum bandwidth of links that the channel passes through. Given the link bandwidth $l_{bw}(n)$ for nth link and the total number of channels $tc(n)$ passing through the nth link, the channel bandwidth $c_{bw}(i, j)$ between sender i and receiver j is calculated using Equation 4.8.

$$c_{bw}(i, j) = min\left(\frac{l_{bw}(n)}{tc(n)}\right) \tag{4.8}$$

In reality, data transferred via a network can be mapped into millions of packets. However, having such a packet modeling approach is difficult, if not impossible, due to memory resource limitations within a software implemented modeller such as IoTSim-SDWAN. To reduce the difficulty of network packet modeling, we use the concept of flow, which is defined as a stream of packets belonging to a given application represented as a 4 tuple ID (source application, destination application, source host, and destination host). For every application between two endpoints, a new flow f must be established. The number of flows f_n between two endpoints is computed using Equation 4.9 where si is the sth application executing on ith node and dj is the dth application executing on jth node. Applications represented by flows share the bandwidth of their assigned channels. We assume the bandwidth is fairly shared amongst the flows. Given the channel bandwidth c_{bw} and number of flow f_n, Equation 4.10 is

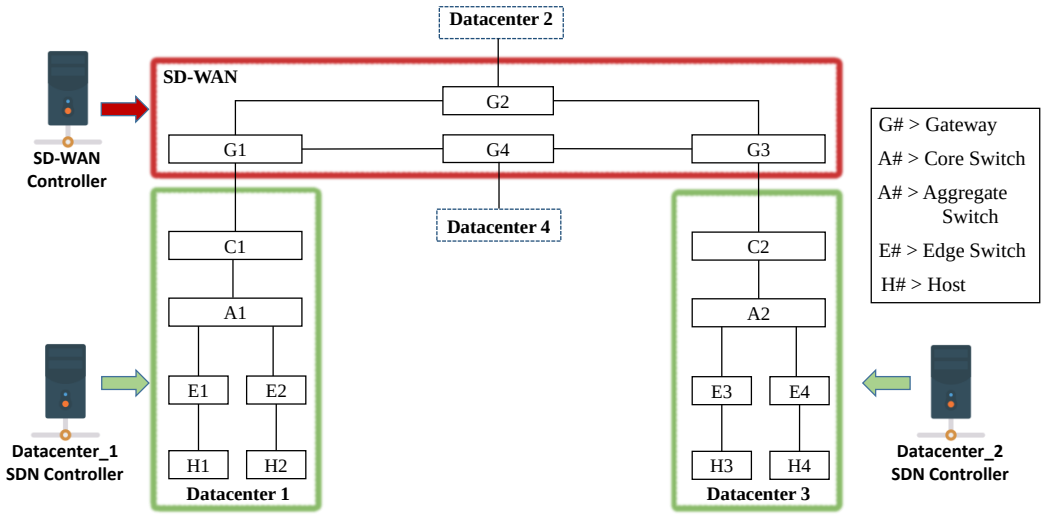


Figure 4.4: Multi-datacenters and SD-WAN topology for experiments.

used to fairly obtain flow bandwidth f_{bw}. The total data size of a given flow f_s is computed by summing the size of all nf frames where the size of each frame is considered to be tf as given in Equation 4.11.

$$f_n = \sum_{f \in F} f(si, dj) \tag{4.9}$$

$$f_{bw} = \frac{c_{bw}}{f_n} \tag{4.10}$$

$$f_s = \sum_{j=1}^{nf} tf(j) \tag{4.11}$$

For every frame, there are transit delays encountered, as shown in figure 4.3. As described earlier in 4.4.1, the main delays are processing d_p, queuing d_q, transmission d_t, and propagation d_ρ. Such delays must be computed for every frame. Equation 4.12 is used to compute all delays d. Equation 4.13 is used to obtain total delays t_d for all frames belonging to a given flow f.

$$d = d_p + d_q + d_t + d_\rho \tag{4.12}$$

$$t_d(f) = nf \times d \tag{4.13}$$

The SD-WAN and SDN-DC are responsible for tracking the transmission time of all its applications/flows. In figure 4.4, every SD-WAN and SDN-DC controller must compute the transmission time for all of generated flows. For instance, the SDN controller in datacenter one (datacenter_1) must track all flows going from and into host one (H1) and host two (H2) up to the core switch (C1). Every network can have different transmission times for every flow due to the fact that the network is shared by other applications. Therefore, the end-to-end transmission time tr of a given flow f passing through nth network can be computed by taking the maximum transmission time of the flow in all networks (SD-WAN and datacenters).

$$tr(f) = \max_{n} \left(\frac{f_s}{f_{bw}} + t_d(f) \right) \tag{4.14}$$

The above equations are implemented in IoTSim-SDWAN . In the validation section 4.5, the equations are filled with numbers according to our real-world network observations using Wireshark [103] and according to TCP/IP predefined header sizes [104].

4.4.3 Network modeling

An SD-WAN simulator must be sufficiently flexible to support different mechanisms that allow changes in experimental contexts (e.g., network topology, QoS) without the need to change the basis of the simulator's actual code. The main flexibly of our tool is allowing different WAN and datacenter topologies, such as the topologies provided by The Internet Topology Zoo [105] where they present hundreds of WAN topologies used by different companies around the globe. For a given topology to be simulated, researchers must code the way that nodes connect to one another along with building internal routing tables, which impedes the researchers to focus on evaluating and solving their intended problems.

One well-accepted solution for solving the aforementioned problem is the use of graph theory. Graph theory is the core solution for network systems to dynamically maintain the location and connection information between nodes. By using graph theory in our tool, we not only analyze and contribute to the performance of SD-WAN and SDN-

Algorithm 1 Shortest Path Maximum Bandwidth

Require: N: a set of nodes (hosts & switches)
f: a network flow containing a stream of packets
s: a source node
d: a destination node
Ensure: f_{route}

/* Construct/update two network graphs, one for distance weight and one for real-time bandwidth capacity */
for each i ∈ N **do**
  for each k ∈ N **do**
    if $distanceWeight[i][k] \equiv \emptyset$ **or** $isNetwrokGraphChange \equiv true$ **then**
      $distanceWeight[i][k] \leftarrow getDistanceWeight(i, k)$ ▷ if i & k adjacent return 1, otherwise return 0
    end if
    /* always update network bandwidth availability*/
    $availabBandwidth[i][k] \leftarrow getAvailableBw(i, k)$ ▷ get real-time bandwidth of a link connecting i & k
  end for
end for
for each n ∈ N **do**
  $distance[n] \leftarrow +\infty$
  $bandwidth[n] \leftarrow -\infty$
  $elected[n] \leftarrow false$
end for
$distance[s] \leftarrow 0$ ▷ distance of source host to itself is always 0
$bandwidth[s] \leftarrow 0$ ▷ bandwidth of source host to itself is always 0
$parentNode[s] \leftarrow \emptyset$ ▷ source node does not have a parent node
for each i ∈ N (we only need N size!!!) **do**
  $minDistance \leftarrow +\infty$
  $maxBandwidth \leftarrow -\infty$
  for each u ∈ N **do**
    if $elected[u] \equiv false$ **and** $distance[u] \leq minDistance$ **and** $bandwidth[u] \geq maxBandwidth$ **then**
      $minDistance \leftarrow distance[u]$
      $maxBandwidth \leftarrow bandwidth[u]$
      $en \leftarrow u$ ▷ en: an elected node with min distance and max bw
    end if
  end for
  $elected[en] \leftarrow true$
  for each k ∈ N **do**
    if $elected[k] \equiv false$ **and** $distanceWeight[en][k] \neq 0$ **and** $distance[en] + distanceWeight[en][k] \leq distance[k]$ **and**
$availabBandwidth[en][k] \geq bandwidth[k]$ **then**
      $distance[k] \leftarrow distance[en] + distanceWeight[en][k]$
      $bandwidth[k] \leftarrow availabBandwidth[en][k]$ ▷ Select the least bw along the route to avoid network congestion
      $parentNode[k] \leftarrow en$
    end if
  end for
end for
$routeBuilt \leftarrow false$
$currentNode \leftarrow d$
$nextNode \leftarrow \emptyset$
while $routeBuilt \equiv false$ **do**
  $nodeLists.add(currentNode)$
  if $currentNode.equal(s)$ **then**
    $routeBuilt \leftarrow true$
  end if
  $nextNode \leftarrow parentNode[currentNode]$
  $link \leftarrow linkList.get(currentNode, nextNode)$
  $routeLinks.add(link)$
end while
$flowLinks.put(f, link)$

DC traffic policies but also to the performance of SD-WAN and SDN-DC routing algorithms. We leverage the classical Dijkstra algorithm [86], which is based on graph theory, for solving the challenge for maintaining a dynamic network graph and finding the shortest path from every node to all other nodes. As figure 4.4 shows, every given

SD-WAN and SDN-DC controller maintains its own network in the form of a sub-graph; therefore, each one must execute its own routing algorithm to properly allocate routes for its respective nodes.

To make our approach more accessible and easier to use, IoTSim-SDWAN has the ability to compare the solutions of traditional WAN and DC networks with the new solutions of SD-WAN and SDN-DC. We simulate classical WAN and DC networks by applying the classical shortest path Dijkstra algorithm (SP) with a single network objective of finding a shortest path based on a minimum number of traversing nodes. Equation 4.15 is used to obtain the objective function of finding a route r that minimizes the traversing number of nodes in traditional networks tn where (n_i, n_j) represents a link connecting two nodes.

$$tn = \min_{n} \sum_{n_i, n_j \in N} r(n_i, n_j) \tag{4.15}$$

We propose a Shortest Path Maximum Bandwidth algorithm (SPMB) to simulate SD-WAN and SDN-DC networks. SPMB is a novel routing algorithm that extends the classical Dijkstra algorithm to obtain a min-max objective. This is designed to find all elected routes that have the minimum number of traversing nodes and then select a route that has maximum bandwidth in real-time. Algorithm 1 shows the pseudo-code of our proposed SPMB algorithm. Equation 4.16 is used to obtain the objective function of finding a route r that minimizes the traversing number of nodes and maximizes bandwidth bw.

$$SPMB = \max_{bw} \left(tn, \sum_{n_i, n_j \in N} r(n_i, n_j) \right) \tag{4.16}$$

Since our proposed algorithm (Algorithm 1) is based on Dijkstra's algorithm with two objectives (Path and Bandwidth) to optimize, the worst case time complexity is $O(N^2+N^2)$, where N represents the number of nodes (hosts and switches). Ultimately, the time complexity is reduced to $O(N^2)$.

Figure 4.5 shows the overview of steps and actions taken by every SD-WAN and SDN-DC controller to enable H1 to send data to H2. In the first step, every controller

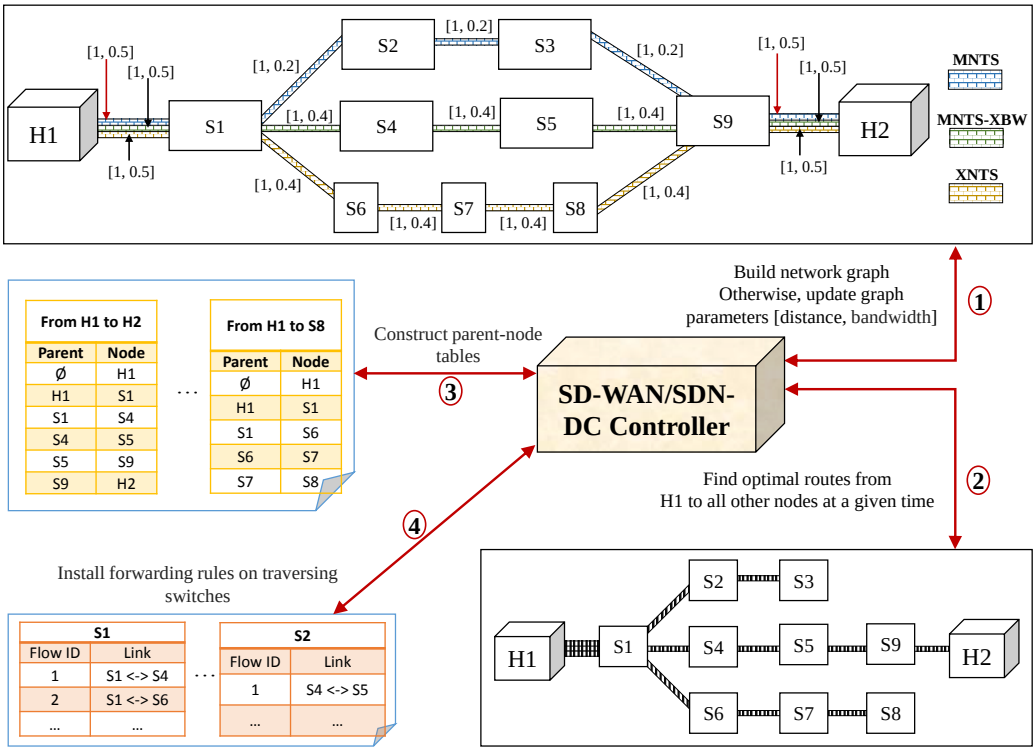


Figure 4.5: Overview of SD-WAN/SDN-DC controller's actions to build forwarding rules. H#: host, S#:switch, MNTS: minimum number of traversing switches, XBW: maximum bandwidth of traversing switches, XNTS: maximum number of traversing switches.

initially builds its own network (sub-graph of the whole network). For every network objective, there must a separate sub-graph. In the case of SPMB, every controller must maintain two sub-graphs, one for storing/updating the minimum number of nodes while the other for storing/updating real-time network bandwidth. As shown, there are two elected routes that obtain the same number of traversing nodes (via S2-S3 and S4-S5). However, once a controller executes its own SPMB algorithm to find the optimal route that has maximum bandwidth between H1 and H2, it only selects the route (H1, S1, S4, S5, S9) that satisfies the objective of SPMB (see step two). In step three, the controller stores the final elected route between H1 and H2 in its routing table. For each child node starting from the destination (H2) up to source (H1), the controller must map the child node to its parent node if it exist. In the final step, the controller installs the forwarding rule on all traversing switches to instruct switches for the appropriate output link/port.

As each SD-WAN and SDN-DC controller separately manages and isolates its network

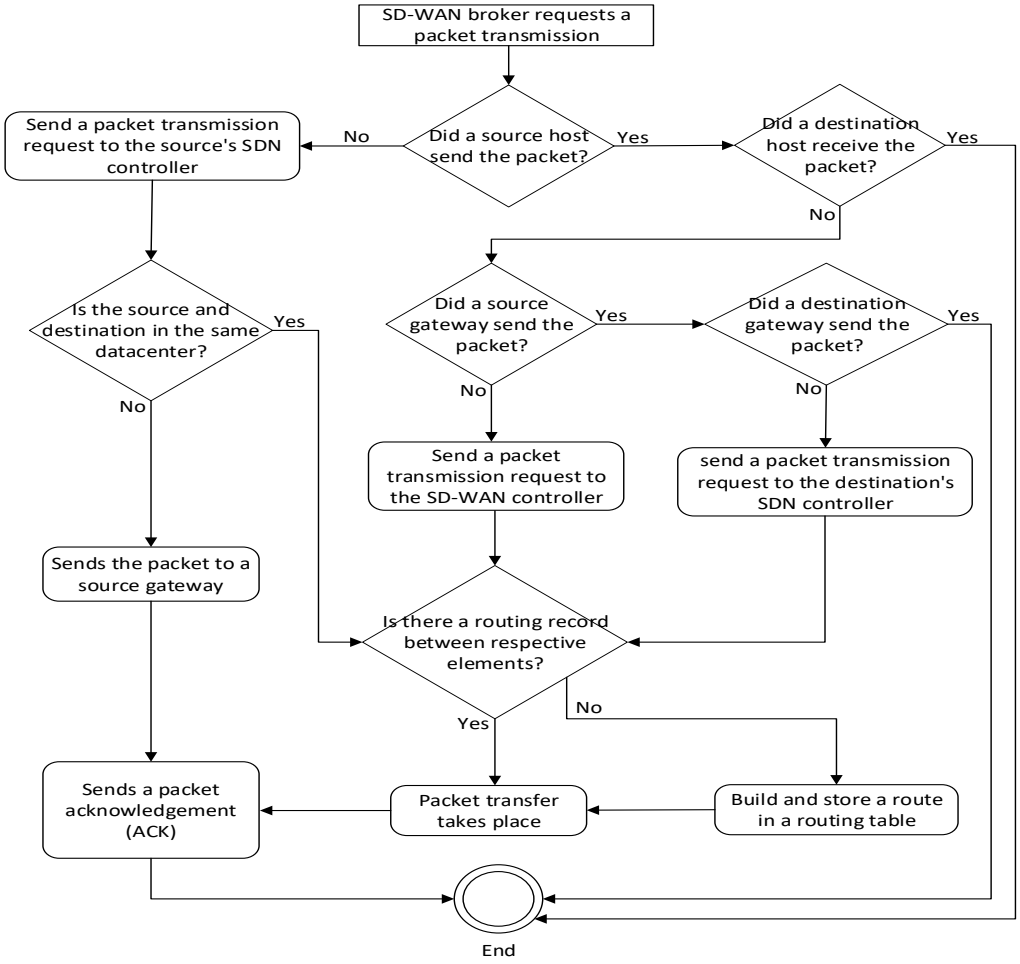


Figure 4.6: SD-WAN Coordination Scheme

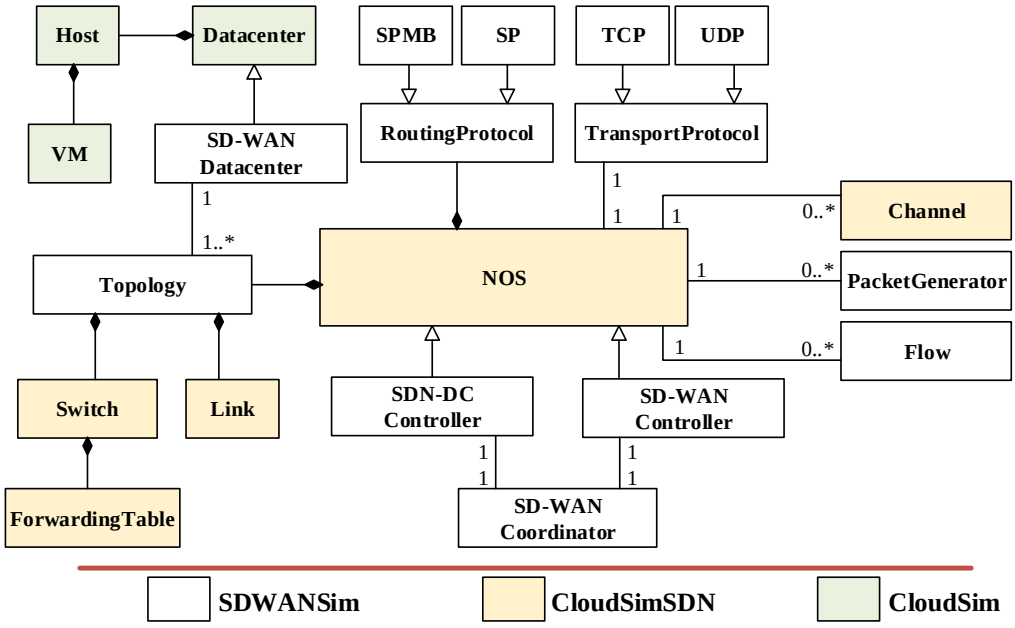


Figure 4.7: System structure of IoTSim-SDWAN

```
"datacentres": [
  {
    "name": "dc1",
    "vmAllocationPolicy": "VmAllocationPolicyCombinedLeastFullFirst",
    "numberOfVms": 2,
    "vmType": 3,
    "controllers": [
      {
        "name": "dc1_sdn1",
        "trafficPolicy": "FairShare",
        "routingPolicy": "ShortestPathBw"
      }
    ],
    "switches":[
      {
        "type" : "gateway",
        "name": "dc1_gateway",
        "controller": "dc1_sdn1",
        "iops":1000000000,
        "bandWidth" : 400000000
      },
      {
        "type" : "edge",
        "name": "edge1",
        "controller": "dc1_sdn1",
        "iops":1000000000,
        "bandWidth" : 200000000
      },
    ],
    "hosts": [
      {
        "name": "host1",
        "pes" : 4,
        "mips" : 1250,
        "ram" : 32750,
        "storage" : 6400000000000,
        "bandWidth" : 200000000
      },
    ],
    "links" : [
      { "source" : "core1" , "destination" : "dc1_gateway" , "latency" : 1.0 },
      { "source" : "core1" , "destination" : "aggregate1" , "latency" : 1.0 },
      { "source" : "aggregate1" , "destination" : "edge1" , "latency" : 1.0 },
      { "source" : "edge1" , "destination" : "host1" , "latency" : 1.0 },
  ]
```

```
  "wan": {
    "wanController": {
      "name": "wan_sdn",
      "trafficPolicy": "FairShare",
      "routingPolicy": "ShortestPathBw"
    },
    "switches":[
    ],
    "wanLinks": [
      { "source" : "dc1_gateway" , "destination" : "dc2_gateway" ,
"latency" : 1.0 },
      { "source" : "dc1_gateway" , "destination" : "dc4_gateway" ,
"latency" : 1.0 },
      { "source" : "dc2_gateway" , "destination" : "dc3_gateway" ,
"latency" : 1.0 },
      { "source" : "dc2_gateway" , "destination" : "dc5_gateway" ,
"latency" : 1.0 },
      { "source" : "dc2_gateway" , "destination" : "dc6_gateway" ,
"latency" : 1.0 },
      { "source" : "dc4_gateway" , "destination" : "dc3_gateway" ,
"latency" : 1.0 },
      { "source" : "dc4_gateway" , "destination" : "dc5_gateway" ,
"latency" : 1.0 },
      { "source" : "dc4_gateway" , "destination" : "dc6_gateway" ,
"latency" : 1.0 },
      { "source" : "dc5_gateway" , "destination" : "dc6_gateway" ,
"latency" : 1.0 }
    ]
}
```


Figure 4.8: An example of JSON input file

(as shown in figure 4.4), they must coordinate with one another in order for each controller to make appropriate (mutually agreeable) routing decisions. Figure 4.6 proposes a simple but effective scheme that is implemented in IoTSim-SDWAN, which enables the the coordination among SD-WAN and SDN-DC controllers. An SD-WAN broker submits network transmission requests on behave of users to respective source SDN-DC controllers. If the source and destination hosts reside in the same datacenter, the transmission process internally takes place and the broker is acknowledged on transmission success. If destination hosts reside in different datacenters, packets will be forwarded to source gateway(s). The transmission process between gateways is handled by an SD-WAN controller. Once the destination gateway(s) receive external packets, they will internally forward the packets to destination hosts. If routing records do not exist, every SD-WAN and SDN-DC controller must execute its routing algorithm and store elected route information in its routing table.

4.4.4 Components modeling and interaction of IoTSim-SDWAN

Figure 4.7 presents a system structure overview of IoTSim-SDWAN. As shown, IoTSim-SDWAN uses some classes of other simulating tools (CloudSim [1] and CloudSimSDN [2]). IoTSim-SDWAN extends the datacenter to enable the interactions with SDN-DC controllers in addition to the provisioning and management of hosts and VMs. SD-WAN and SDN-DC controllers extend the class of networking operating systems (NOS) where the designs, functionalities, and characteristics of such controllers are implemented. The SD-WAN coordinator is responsible for sharing and advertising the required information among SD-WAN controllers, SDN-DC controllers, and datacenters; for example, sharing the location of requested hosts in order to build appropriate routes to respective destinations.

IoTSim-SDWAN describes the arrangement and relation among components in the topology class. The topological description is provided to IoTSim-SDWAN in a JSON file format. Once the file is submitted, IoTSim-SDWAN will instruct the topology class to parse, generate, and store the properties and relations among components. Figure 4.8 shows an example of the JSON format. As shown, every datacenter must define and configure its own topology that includes SDN-DC controllers, switches, and hosts in addition to the links that connect switches and hosts to one another. In addition, an SD-WAN topology must be defined and configured in terms of specifying the properties of SD-WAN controllers and links that connect datacenters together.

Every SD-WAN and SDN-DC controller must implement routing protocols/algorithms. The routing protocol class is designed to facilitate the implementation of such algorithms by providing abstract functions that can be used to develop smarter routing algorithms. Currently, IoTSim-SDWAN contains two routing algorithms (SPMB and SP) as described in sub-section 4.4.3. New routing algorithms can easily extend the routing protocol class. IoTSim-SDWAN couples every switch with a forwarding table so that controllers can manipulate the way switches forward packets. Figure 4.9 illustrates how controllers handle network packets. Every incoming packet must be put into a queue so that controllers may serve packets in order. Packets will obtain their optimal routes dictated by the routing algorithms of controllers.

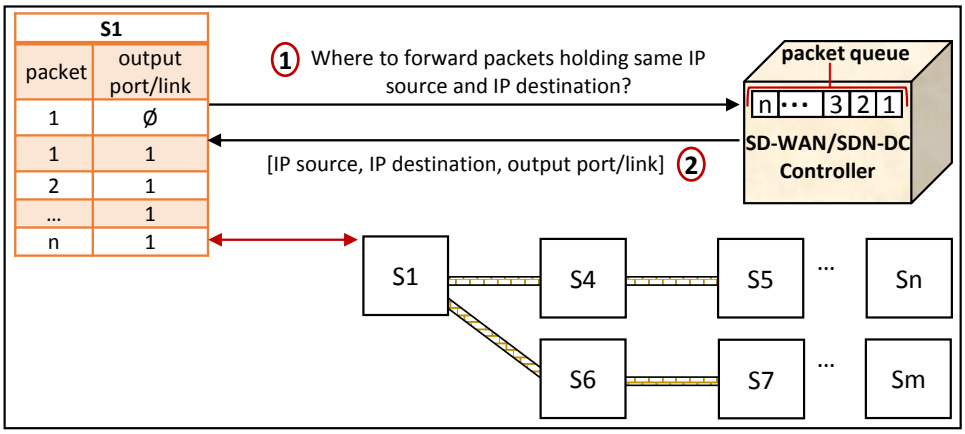


Figure 4.9: An interaction between OpenFlow switches (S#) and SDN for obtaining flow entry

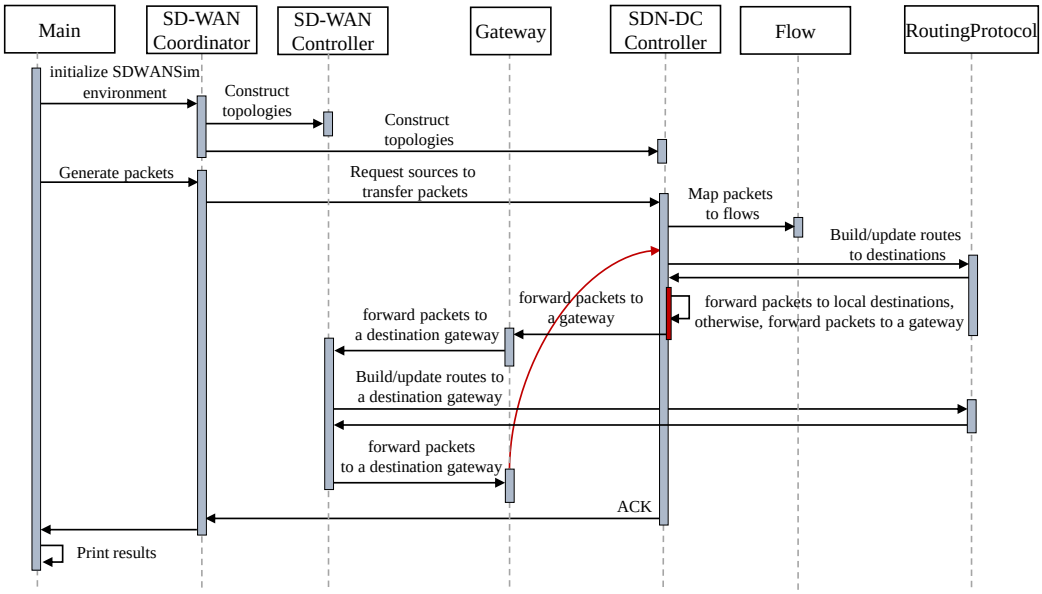


Figure 4.10: Underlying interactions of IoTSim-SDWAN

For IoTSim-SDWAN to function correctly, the transport protocol must be specified. IoTSim-SDWAN is equipped with two transport protocols: TCP and UDP. A given SD-WAN and SDN-DC controller must be instructed on which protocol is to be utilised. This allows computation of overhead bytes or headers and footers for generated packets in a given network. Moreover, the header data introduced by network and data link layers must also be computed. In the validation section 4.5, the number of bytes to be added to the original data are presented according to our real-world network analysis and observation using the Wireshark network monitoring tool [103].

Figure 4.10 illustrates the interaction across the components of IoTSim-SDWAN in

a simplified form. The generation of packets will take place once IoTSim-SDWAN initializes the required infrastructure according to a submitted JSON file. Generally, the life-cycle of SD-WAN packets start from a source datacenter, pass through an SD-WAN network, and end in a destination datacenter (see figure 4.4). The submission of packets is carried out by a broker who forwards packets to their respective source SDN-DC controllers. When SDN-DC controllers receive packet transmission requests, they will correlate packets to flows, find optimal routes using provided routing protocols/algorithms, and instruct switches to forward packets to destination hosts. If destination hosts are not within a given datacenter, packets will be forwarded to a gateway switch. Once the packet reaches the gateway, it is routed to the appropriate datacenter using an SD-WAN controller. When packets reach the gateway of a destination datacenter, an SDN-DC controller residing in the destination datacenter will find the appropriate route to local destination hosts and acknowledge/report back the output results once packets reach destination hosts.

4.5 Empirical Validation of IoTSim-SDWAN

Validating IoTSim-SDWAN against real-world networks is crucial in order to illustrate its accuracy and efficacy and prove its models produce results that are realistic and reflective of existing systems. It is worth noting that reaching maximum network capacity for a given real-world network environment is difficult to achieve (and unwarranted for live systems due to service disruption). Therefore, to benchmark at this extreme is difficult to achieve. Furthermore, network and system engineers may identify suitable factors that effect network capability and may inform a model (e.g., identifying network protocols that consume a part of the network bandwidth), but may not be able to identify other hidden factors (e.g.,how receiving hosts queue and deal with network packets and perform read/write operations on hard drives). Therefore, accepting a slight difference between theoretical models and reality is acceptable when measuring real network environments with the theoretical output values. A slight difference occurring in IoTSim-SDWAN should, therefore, also be acceptable. Nevertheless, we have tried to eliminate the difference rate of IoTSim-SDWAN compared with real networks by understanding and observing the conditions and behaviours that

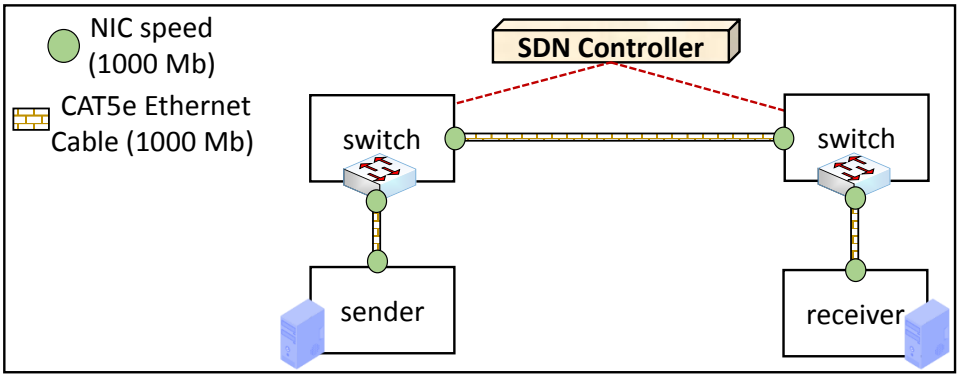


Figure 4.11: Network topology design for the validation experiments

effect the performance of real network systems. Based on our observations, we have derived and established the theoretical model in sub-section 4.4.2. The validation results proves that IoTSim-SDWAN is capable of obtaining a high degree of accuracy compared with real networks.

4.5.1 Validation setup and configuration

The validation experiments are conducted on two machines that have Intel Core i7-7500U 2.70GHz and 16GB of RAM memory. Each machine has an installed guest host (VM) running Linux 4.4.0-31-generic. Each VM is configured with 4 virtual processors and 4GB of memory. Two Linux-based switches designed by Shenzhen Helor Cloud Computer [106] are used with similar configurations. Every switch has Intel Celeron 1037U (2 Cores, 1.80GHz), 4GB of memory, and 6 Ethernet ports each attaining 1000Mbps of throughput. Every switch is installed with an OpenFlow switch (OVS) [107], which enables controllers to instruct and manipulate the data plane of switches. The SDN controller used is Ryu SDN framework [108]. Machines and switches are connected via Ethernet cables Cat5e with 1000Mbps of speed.

For validating IoTSim-SDWAN, a number of real and simulated experiments are carried out. Validation objectives are to:

- Identify the correctness, accuracy, and credibility of the IoTSim-SDWAN framework compared with a real-world network environment in terms of bandwidth, network transmission time, TCP/UDP outputs, and network delays.

- Measure how realistically a real-world network can reach its maximum network bandwidth.
- Observe the internal impact of applications and system structures of hosts/servers (CPU, memory, and hard drives) on network bandwidth/speed.
- Validate TCP and UDP models of IoTSim-SDWAN compared with a real-world network environment.

Figure 4.11 shows a network topology design for the validation experiments. Similar designs are also developed for IoTSim-SDWAN. The validation is carried out using three different types of applications: Iperf3 TCP, Iperf3 UDP, and Secure Shell (SSH). All of the experiments have the Internet connection disabled and only run intended applications, excluding default applications and those operations required by operating systems. In this step, unintentional use of networks and operating systems by unintended applications is eliminated. Iperf3 [109] is a well-known network tool for capturing and analyzing network performance in terms of TCP and UDP protocols. It is mainly designed to measure network throughput, which means that the internal structures of given hosts have no or minimal impact on network performance. To ensure we consider internal structures, SSH is used to transfer a video file between hosts that involves the internal structures of hosts (e.g., memory, CPU, and hard drives). By using SSH, the impact of hosts' internal structures on network performance can be captured, which allows further validation of IoTSim-SDWAN.

The configuration used to validate IoTSim-SDWAN is given in Table 4.3. The header sizes and average payloads of given applications are obtained according to our Wire-

Table 4.3: Validation configuration

TCP		**UDP**		**delays**	
Transport header size	20 bytes	Transport header size	10 bytes	Processing	0.001 ms
Network header size	20 bytes	Network header size	12 bytes	Queuing	0.001 ms
Data link header size	26 bytes	Data link header size	12 bytes	Propagation	0.0001 ms
-	-	-	-	Transmission	0.0001 ms

App	**data sizes**	**Average packet payload**	**Average frame payload**
Iperf3 UDP	1061.49 MB	8146 Bytes	1366 Bytes
Iperf3 TCP	778.35 MB	21464.8 Bytes	1448 Bytes
Video file	493.42 MB	13007 Bytes	1408 Bytes

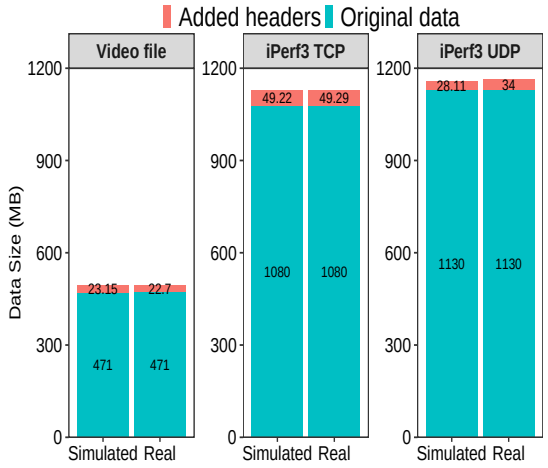
shark observations. Capturing every single delay of real network processing, queuing, transmission, and propagation is beyond the scope of this chapter. However, obtaining the total delay of a real network environment is possible using a time counter. Delays are considered to be in the order of milliseconds [110].

The management layer of the SDN is added to our empirical network to program, manage, and instruct our network during execution (see figure 4.9). The SDN mechanism does not change a network's built-in capabilities, such as network bandwidth, but does manage the network in real-time (e.g., finding global optimal routes between given nodes, reserving a part of network bandwidth for given applications). The interaction time between OpenFlow switches and the SDN controller is negligible and does not affect the overall network performance. As it can be seen in figure 4.9, an OpenFlow switch requires a one-time reactive interaction with the SDN controller to obtain a flow entry for a stream of packets holding same IP source and IP destination. The SDN controller, for example, would determine the best path for the given flow entry and feed the switch (s1) with routing information. The controller can also instruct the switch to hold the flow entry for a period of time (e.g., 30 seconds), which means that the switch knows how to forward the remaining stream of packets based on the given time to live

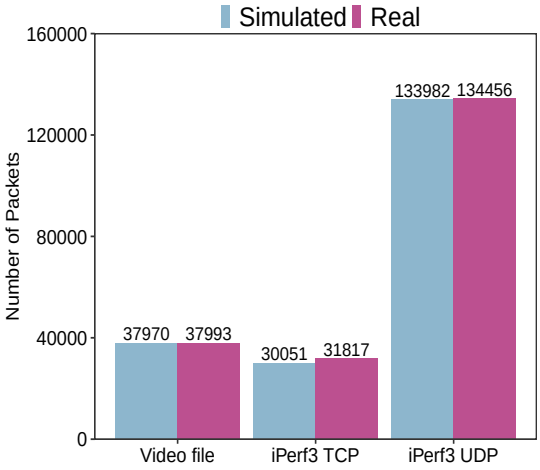
4.5.2 Validation results

Figure 4.12 shows the overall header data sizes added to original data. The original data size of each application is increased when being transferred due to the added network headers. Each header size is injected by each of its respective layer as shown in the network model (see figure 4.2). We can see that IoTSim-SDWAN is capable of obtaining approximate header sizes compared with the real-world network environment. In Iperf3 UDP, however, simulated header sizes have a slight difference due to the average header size not always reflecting expected accuracy.

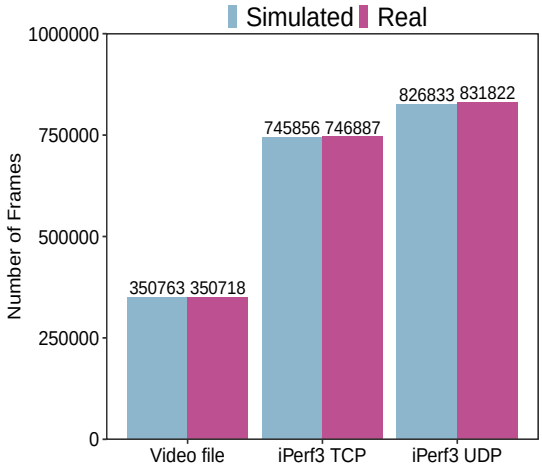
Figure 4.12b and 4.12c show the the number of packets and frames. We can observe that the number of packets and frames of each application in IoTSim-SDWAN and the real environment is closely comparable. The average payload factors of packets and frames as given in 4.4.2 verify that we can obtain approximate results as compared to



(a) Size of network headers added to original data



(b) Comparison of number of packets

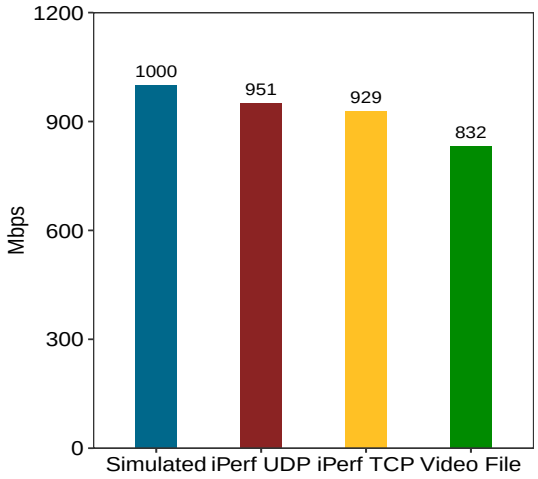


(c) Comparison of number of frames

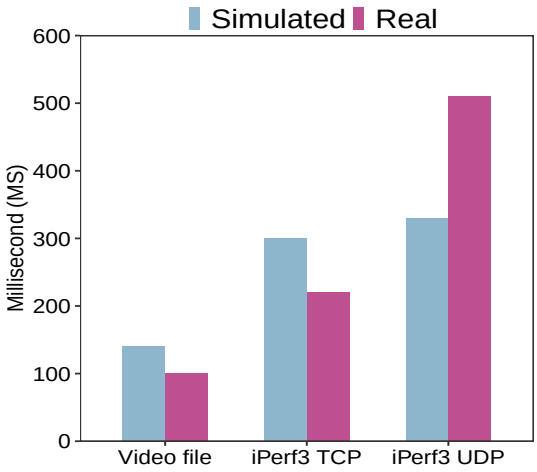
Figure 4.12: Comparison of IoTSim-SDWAN and a real network environment

real network environments, even though the distribution payload sizes of packets and frames may vary in real network environments.

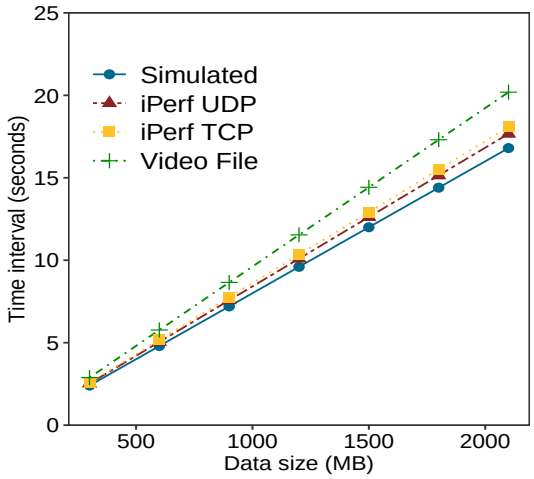
Figure 4.13 shows the comparison of bandwidth, speed, and delay between IoTSim-SDWAN and a real network environment. In figure 4.13a, the maximum bandwidth rate of Iperf3 UDP is 951Mbps while for Iperf3 TCP is 929Mbps. Iperf3 cannot achieve the speed of the theoretical network bandwidth (1000Mbps) because there are additional factors affecting Iperf3, such as the performance of the network interface cards (NICs) of hosts and switches, CPU performance of hosts and switches. The bandwidth for the video file is 832Mbps, which is less than Iperf3 bandwidth. Unfortunately, transmitting real data (e.g., video files) not only depends on network performance but



(a) Comparison of bandwidth



(b) Comparison of delays



(c) Comparison of transmission time

Figure 4.13: Bandwidth, speed, and delay comparison of IoTSim-SDWAN and a real network environment

also on the performance of given hosts in terms of CPU, memory, and hard drives. Iperf3 generates random data originating from CPU (no hard drive) while the video file application requires data from a hard drive and then writes the data to memory, which degrades the overall performance.

Figure 4.13b shows overall network delay as a result of processing, queuing, transmission, and propagation delays. We can observe that the delays of IoTSim-SDWAN and the real network are closely comparable. Figure 4.13c compares the IoTSim-SDWAN with the real network environment in terms of transmission time. It can be seen that IoTSim-SDWAN and the real network environment have a positive correlation. The transmission time of IoTSim-SDWAN compared with iperf3 TCP and iperf3 UDP is

Table 4.4: Evaluation configuration

Sender	Receiver	Protocol	data sizes	Average packet payload	Average frame payload
Host 1 (H1)	Host 3 (H3)	TCP	10 Gb	21464.8 Bytes	1448 Bytes
Host 2 (H2)	Host 4 (H4)	TCP	10 Gb	8146 Bytes	1366 Bytes

closely comparable. IoTSim-SDWAN and the video file has a slight difference in transmission time, which is expected since the video file depends on the performance of the internal structures of hosts in addition to the network. As IoTSim-SDWAN is mainly intended to simulate the network layer and components of SD-WAN; therefore, iperf3 is the suitable candidate to validate the accuracy of IoTSim-SDWAN.

4.6 Use case evaluation

This section is intended to demonstrate the practicality and advantages of IoTSim-SDWAN. Mainly, we consider the comparison of performance and energy-consumption of SD-WAN and classical WAN environments in addition to traditional cloud datacenters compared with SDN-aware cloud datacenters. Figure 4.14 presents the network topology design used in the evaluation experiments. Both SD-WAN and classical WAN have a similar topology design. Every datacenter has a single gateway that is connected to other datacenter gateways. In the classical WAN and cloud datacenters, gateways and switches have full control of their network decisions. In SD-WAN and SDN-aware environments, SD-WAN and SDN controllers have full network control to instruct gateways and switches to enable the management and influence on network traffic in real-time. Table 4.4 shows the configuration used in the evaluation experiments. Using TCP or UDP protocol would not impact the evaluation results; therefore, TCP is the protocol that we use in this evaluation. The average payload size of packets and frames in addition to header sizes and delays are similar to the ones used in the validation section in table 4.3.

IoTSim-SDWAN shapes network traffic in SD-WAN and classical WAN environments according to the network model in 4.4.3. As mentioned earlier, the classical shortest path Dijkstra algorithm (SP) is used to shape network traffic in classical WAN environments whereas SD-WAN shapes its traffic based on our proposed SPMB algorithm (see algorithm 1). Figure 4.14 conceptually illustrates how the SD-WAN and classical

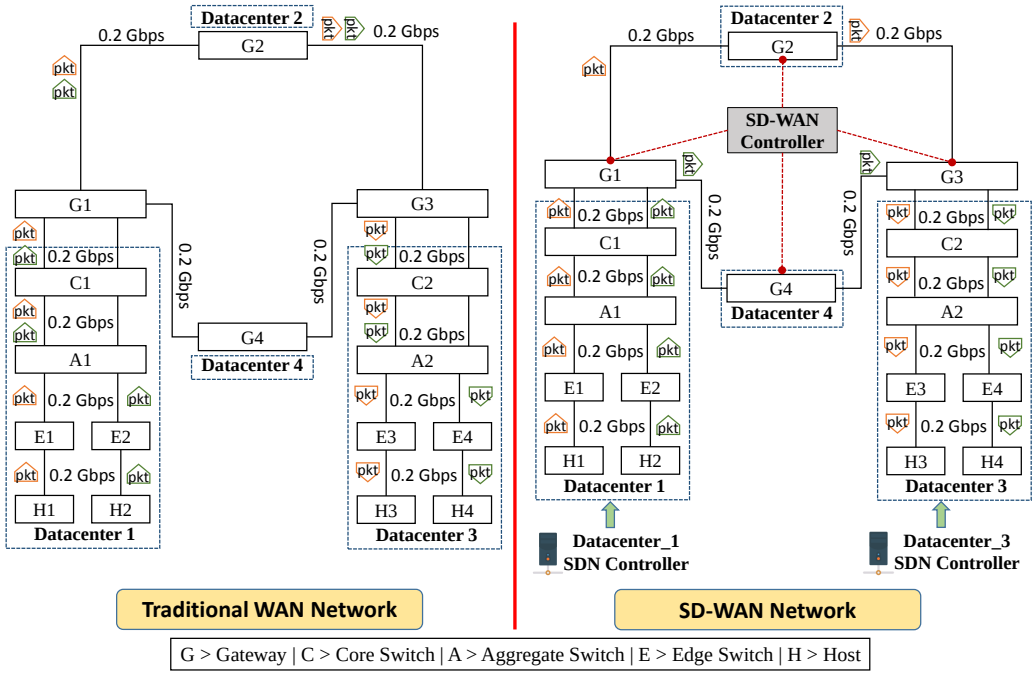
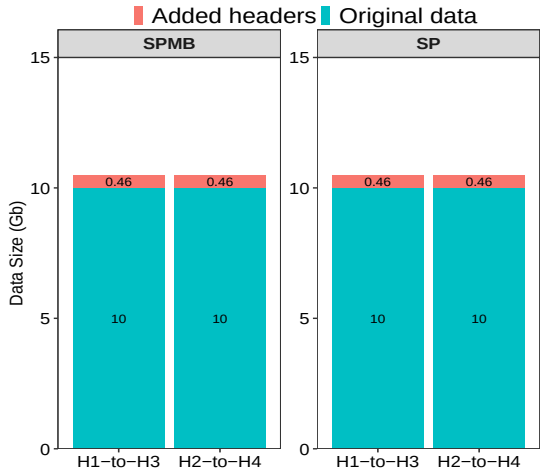


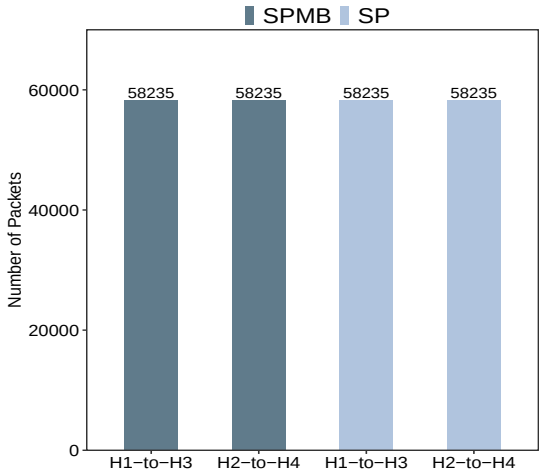
Figure 4.14: Network topology used in IoTSim-SDWAN

WAN work. Both environments must forward packets that are generated from H1 and H2 residing in datacenter 1 to H3 and H4 residing in datacenter 3. In the classical WAN, the packets have the same route where they fairly share network bandwidth. This is because the classical WAN lacks the ability to dynamically forward packets based on real-time changes in network congestion and bandwidth availability. However, SD-WAN is capable of obtaining such information and forwards the packets to different routes based on appropriate choice. The coordination of SD-WAN and SDN controllers is also possible to efficiently improve their traffic engineering decisions, as shown in figure 4.6.

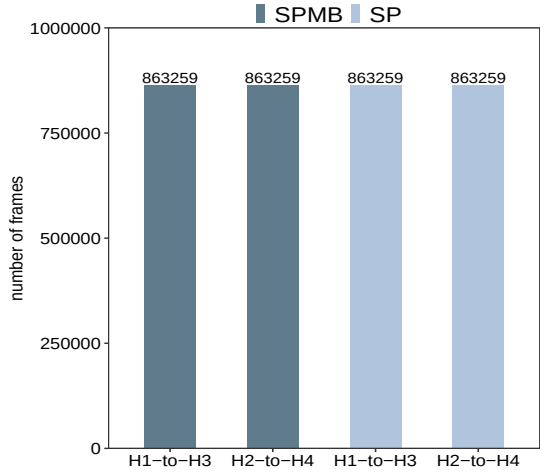
Figure 4.15 shows the header data sizes added to the original data, number of packets, number of frames, and end-to-end delays. It can be seen that the SD-WAN (SPMB) and WAN (SP) have the same number of header sizes, packets, frames, and delays. This is expected because both SD-WAN and WAN have no impact on TCP and UDP protocols. To appropriately evaluate the network performance of SD-WAN and WAN, both networks must be overwhelmed with packets from different sources at the same time. Packets from sources to destinations are being transferred simultaneously during the simulation. Figure 4.16a illustrates the network transmission time of SD-WAN



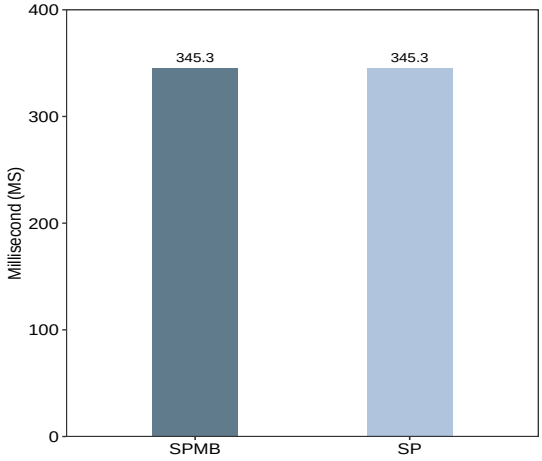
(a) Size of network headers added to original data



(b) Comparison of number of packets



(c) Comparison of number of frames

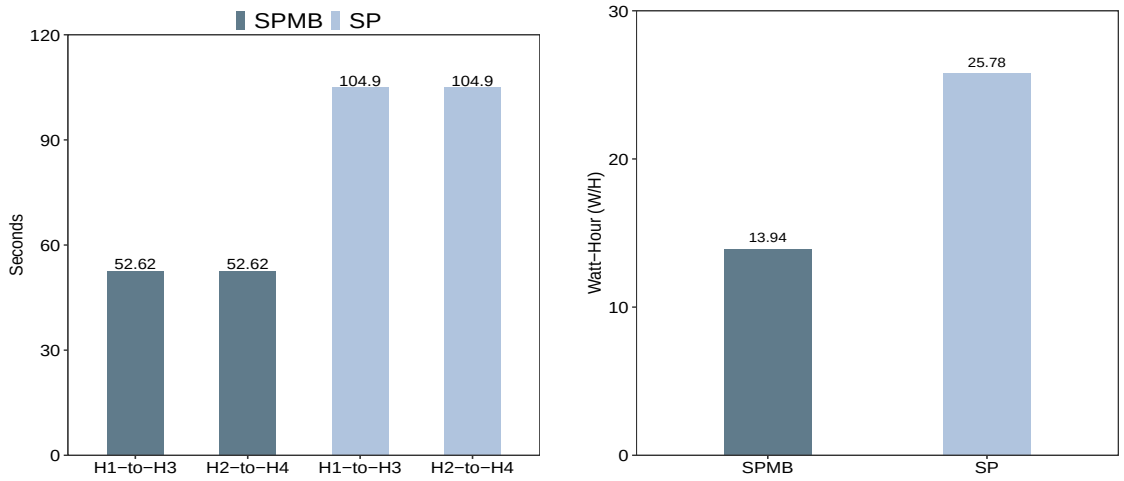


(d) End-to-end delays

Figure 4.15: Simulation result and comparison between SPMB and SP

and WAN. It can be seen that SD-WAN optimized network performance and usage by decreasing the transmission time by approximately 50%. Exceptional performance is achieved because SD-WAN and SDN-DC controllers are capable of locating the least congested routes in addition to minimizing the number of traversing nodes in real time, while the classical WAN can only find routes in a static manner (determined in advance in most cases).

Further, we illustrate the practicality of IoTSim-SDWAN by using energy consumption models. These allow the gathering and reporting of power consumption of switches and gateways. The modeling of power consumption is done by CloudSimSDN [2]. Figure 4.16b shows the overall energy consumption of both WAN and SD-WAN environments.



(a) Data transmission time (b) Energy consumption

Figure 4.16: Simulation result and comparison between SPMB and SP

As the figure shows, SD-WAN consumes less energy than WAN by approximately 54%. The reason SD-WAN consumes less energy is that packet transmission is faster and if packets spend less time within the infrastructure then fewer resources are consumed.

4.7 Conclusion

In this chapter, we present a new tool IoTSim-SDWAN for simulating the behavior and properties of SD-WAN and SDN-aware cloud datacenters. IoTSim-SDWAN is a new tool providing a variety of modeling approaches and functionalities to evaluate and test SD-WAN cloud-based solutions. IoTSim-SDWAN obtains several models, including SD-WAN ecosystems, TCP and UDP protocols, network delays, in addition to the network routing models of SD-WAN and classical WAN using graph theory. We propose a coordination scheme for SD-WAN and SDN controllers residing in different datacenters.

We empirically validate and analyze the accuracy and correctness of IoTSim-SDWAN. Three different types of experiments are used in the validation: Iperf3 TCP, Iperf3 UDP, and transferring real data over Ubuntu Secure Shell (SSH). The validation considers measuring the level of similarities of IoTSim-SDWAN and a real-world network environment in terms of bandwidth, transmission time, TCP/UDP outputs, and network delays. The validation results prove that the accuracy and correctness of

IoTSim-SDWAN are closely comparable to real networks.

We model and present a number of evaluation experiments with a goal to illustrate the practicality and features of IoTSim-SDWAN. The evaluation compares the network performance and power consumption of SD-WAN and classical WAN. The evaluation results demonstrate that SD-WAN outperforms WAN in both performance and energy consumption. Such evaluation experiments also provide a flexible approach for designing new experiments in order to help researchers to seamlessly implement and evaluate their new SD-WAN, SDN routing and power consumption approaches.

Software availability: The IoTSim-SDWAN's software with the source code can be downloaded from https://github.com/kalwasel/IoTSim-SDWAN. A number of examples and tutorials illustrating the use of IoTSim-SDWAN are given on the web site.

5

A Green Energy-Aware Scheduling Algorithm for Data-Driven Applications in Software-defined Multi-Cloud Environments

Contents

Summary

This chapter proposes an adaptive solution for scheduling the workflows of data-driven applications across multiple SD-WAN-enabled cloud datacenters. It does so by leveraging a well-known metaheuristic genetic algorithm and SD-WAN networks that smartly interconnect cloud datacenters to one another. The proposed solution is capable of increasing the usage of green energy which reduces carbon footprints and at the same time meet user given deadline and keeps energy cost to a minimum. The performance of the proposed algorithm is evaluated using real-world data-driven workflows with different sizes under various configurations. The experimental results show that our proposed method can increase the usage of green energy up to 3× times with a slight increase in energy cost when compared to cost-based workflow scheduling methods.

5.1 Introduction

Data-driven applications have emerged to improve the processes and operations of numerous fields (e.g., smart cities, Industry 4.0) based on the leveraging of a range of smart technologies. Cloud computing is considered one of the main building blocks that effectively provide on-demand infrastructures to facilitate data-driven workflows [111, 112]. As cloud computing has proved its capability and success for providing technological needs on-demand and with low-cost prices, data-driven applications started to deploy their complex workloads and elements on remote clouds. As such, this leads to an increase in data-driven workflows forwarded and executed at cloud datacenters, which requires new scheduling mechanisms to satisfy a range of data-driven applications' requirements.

It is quite evident that the amount of energy consumed by datacenters worldwide is continuously increasing, which puts an enormous strain on energy sources. In 2010, datacenters had consumed about 1.3 percent of total electricity usage worldwide [113]. Since then, the datacenter landscape's size has increased dramatically, which claimed to be doubled in their electricity usage worldwide [114]. Clearly, to operate such large infrastructures, a tremendous amount of electricity is required, subject to a given datacenter's size. Such increase in electricity usage not only contributes to high operational

costs, but also emits considerable amounts of carbon dioxide (CO2). According to a report published by the European Union,[1] the volume of emission worldwide must be decreased to keep the global temperature below 2°C. Therefore, the green/renewable energy procurement and energy usage optimization of cloud datacenters are essential elements to decrease the level of human-caused global warming.

In order to minimize non-green energy usage while avoiding SLA violations of data-driven applications, new system and algorithmic solutions are required. The solutions should consider several factors, including the topology and deadline of data-driven tasks, green energy usage, leveraging SD-WAN capabilities, and decreasing overall energy consumption and cost. Such solutions require cross-layer optimization techniques, which are defined as NP-hard problems [115]. To this end, this chapter proposes an adaptive genetic algorithm to schedule the workflows of data-driven applications considering these factors, in addition to the use of the previously proposed SPMB algorithm to effectively transfer data-driven traffic among distributed datacenters through SD-WAN, referred to Chapter 4. In particular, our proposed algorithm minimizes execution cost while selecting solutions that minimize carbon footprints by using multiple datacenters with more green energy usage and interconnected via SD-WAN (Table 5.3 highlights the novelty of our work). The contributions of this chapter are:

- A new SD-WAN based Workflow Broker (SDWAN-WB) to deploy data-driven workflows across multiple datacenters while automating resource provisioning, task provisioning, and data provisioning.
- An adaptive Genetic Algorithm (GA) for selecting near-optimal solutions based on green energy usage, the topology and deadline of data-driven tasks, and overall energy consumption and cost.
- Leveraging SD-WAN capabilities to decrease overall network energy consumption.
- A trade-off analysis of different factors (e.g., energy cost, green energy availability, data-driven requirements) and extensive experimental evaluation to study

[1] https://www.ec.europa.eu/clima/policies/strategies/progress$_e n$

the feasibility of the proposed scheduling algorithm and architecture based on real data.

The rest of the chapter is organized as follows. Section 5.2 highlights the research problems addressed by this chapter. Next, we present our proposed solutions (SDWAN-WB, system model, and GA algorithm) in Sections 5.3 and 5.4. In Section 5.5 and 5.6, we evaluate the experimental results. Section 5.7 describes related work by comparing the chapter's proposed solutions with state-of-art efforts. Section 5.8 provides concluding comments and future work.

5.2 Overview

5.2.1 Executing data-driven applications across multiple datacenters via SD-WAN

To execute the workflows of data-driven applications on cloud datacenters, we need sufficient computing resources, the code-base of data-driven tasks, and actual data to be processed (e.g., provided by users, generated from upstream tasks). To this end, the scheduler of data-driven applications have to handle three factors at the same time: resource provisioning, task provisioning, and data provisioning. Some studies [116, 117] developed schedulers to run scientific workflows over multiple datacenters. However, they did not embrace the SD-WAN capabilities, which offers more flexibility for designing new green energy scheduling algorithms. Another study [118] proposed a method that leverages SDN capabilities to optimize the latency of tasks in the cloud network and VM layers, in addition to maximizing the revenue of cloud providers. However, this solution does not consider SD-WAN capabilities to interconnect multiple cloud datacenters along with optimizing overall network performance and energy consumption.

5.2.2 Optimizing energy efficiency while avoiding SLA violations

While cloud computing aims to optimize the use of hosted Information and Communication Technology (ICT) resources, cloud providers should have an effective solution

for simultaneously optimizing energy consumption and SLAs (e.g., deadline, processing cost), especially for data-driven applications in software-defined geo-distributed environments. One primary obstacle for obtaining such solutions is that cloud providers operate multiple large datacenters distributed across multiple locations. Depending on the location of datacenters and applications, the scheduling process for every application has to automate the cloud datacenter selection and, in doing so, ensure that SLA and energy requirements (e.g., task deadline, total electricity bills, sustainability goals) are met at the same time, which are often conflicting. When selecting ICT resources (e.g., VMs, containers, storage) from multiple datacenters, cloud providers must consider a heterogeneous set of criteria and complex dependencies across multiple layers (e.g., application level, datacenter level) which is impossible to resolve manually.

There is a substantial amount of related work for reducing carbon footprints of datacenters by managing customers' workloads at different levels, such as storage, computation, and network [119–121]. However, most of these solutions are not directly applicable in the context of data-driven applications and SD-WAN, which is the focus of this chapter. As each application has a different workload and execution profile, a unique strategy is needed to minimize energy usage while optimizing SLAs. Data-driven applications are among the most complex applications where several tasks have to be executed in a synchronous and inter-connected manner to achieve the required QoS [122]. Communication between different tasks makes the matter worse as network's energy usage also needs to be considered, along with other constraints. Several authors have proposed several solutions that embrace the idea of distributing given tasks across multiple datacenters to improve energy cost and minimize energy environmental impacts [123]. However, these solutions are designed for simple applications that consist of independent tasks. Also, the existing solutions do not embrace the advantages of SD-WAN capabilities to improve data provisioning and energy consumption. A comprehensive comparison of the chapter's proposed work with the state-of-the-arts is discussed later in Section 5.7.

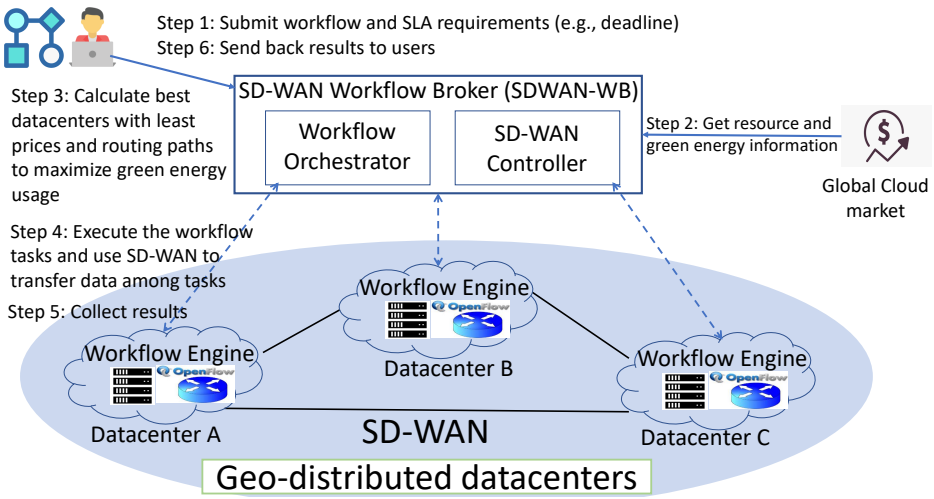


Figure 5.1: High level system model for executing data-driven applications

5.3 System model

Figure 5.1 presents a high-level of our proposed system model, which contains the components of SDWAN-WB. The components are used for executing data-driven applications over geo-distributed cloud datacenters in an efficient manner. The proposed system $S = (dc_k \cup sd_g \mid k \in \{1, \delta\}, g \in \{1, \zeta\})$ consists of a set datacenters (DC) owned by a cloud provider (e.g., Amazon EC2) where $dc_k = \{vm_i \in \mathcal{VM}, d \in D, n \in N\}$ represents a datacenter k which contains a set of VMs, storage (D), and network devices (N). $sd_q \in Q$ denotes an SD-WAN-enabled network devices that interconnect datacenters to each other.

In the figure, a cloud provider utilizes the *workflow orchestrator* to deploy the workflows of data-driven applications across multiple datacenters, and the *SD-WAN controller* is used to optimize data transmission among geo-distributed datacenters so that a given data-driven application is executed with minimal execution cost and carbon footprints. In particular, users submit their workflows and application logic with all information such as execution requirements, task descriptions, and desired security requirements to our broker. The workflow orchestrator is responsible for mapping workflow tasks to different datacenters based on electricity prices and green energy usage, along with meeting other constraints (e.g., deadlines). Based on the workflow orchestrator's decisions, it interacts with each datacenter to prepare VMs to execute the workflow tasks according to defined dependencies/orders. The SD-WAN controller

manages data transmission among different tasks during the execution by dynamically configuring the forwarding-tables of the related SD-WAN-enabled network devices.

5.3.1 User's cost model

We assume that each datacenter dc_k has three types of VMs: $Type(vm_i) \in \{small, medium, large\}$. This chapter assumes that users are charged according to how long they use VMs. Based on this assumption, the cost of executing a workflow task $a_m \in \mathcal{A}$ over $vm_i^m \in \mathcal{VM}$ can be computed using Equation 5.1 where $\mathcal{A}$ is a set of activities (e.g., processing, transmission) that belongs to a given workflow λ. $T_l(vm_i^m)$ is the time required to launch a VM i that host a workflow task m, $T_e(vm_i^m)$ represents the time required to execute a_m over vm_i, $VPrice(Type(vm_i^m))$ denotes the price of a selected vm_i according to its type, $T_t(vm_i^m)$ denotes the time required to transfer a_m's input data to vm_i, and $NPrice(vm_i^m)$ represents the network usage price based on a_m's data size. If all the input data are in the same VM, the network transmission time and cost is equal to 0 (e.g., $T_t(vm_i^m) = 0$, $NPrice(vm_i^m) = 0$). The models for calculating these times will be detailed in the next subsection.

$$\begin{aligned} UCost(a_m, vm_i) =& (T_l(vm_i^m) + T_e(vm_i^m)) \times VPrice(Type(vm_i^m)) + \\ & T_t(vm_i^m) \times NPrice(vm_i^m) \end{aligned} \tag{5.1}$$

5.3.2 Performance model

The performance (makespan) of executing the workflow of a data-driven application over different VMs that are deployed across different datacenters includes three elements: the time a VM (T_l) takes to be launched, the network transmission time of task a_m's input data to its allocated VM (T_t), and the execution time (T_e) of a workflow task a_m. In order to model this, we assume that the launching time of each type of VM is the same, which is equal to T_l. Next, network bandwidth is denoted as $tp_{vm_i} \in \{tp, 2tp, 3tp\}$, which respectively corresponds to $Type(vm_i) \in \{small, medium, large\}$. Moreover, data transmission time is not only affected by VMs' bandwidth, but also by the VMs' geographical locations and the number of traversing

network devices. The following illustrates how data transmission time is formulated.

Single datacenter. A workflow can have numerous dependant tasks. For example, assume a single workflow has two dependant tasks (a_1 and a_2) where data is generated from a_1 and transferred to a_2. The tasks are allocated into two different VMs (vm_1 and vm_2) in the same datacenter. As such, the network bandwidth between a_1 and a_2 can be acquired as $tp_{a_1 \rightarrow a_2} = min(tp_{vm_1}, tp_{vm_2})$. So, the time required to transfer P size of data from a_m to a_j is computed as $T_t(a_m, a_j) = \frac{P}{min(tp_{vm_i^m}, tp_{vm_x^j})} + P \times \varphi \times H(dc_k^{m,j})$ where vm_i^m is the ith VM that hosts a workflow task a_m, φ is the average latency incurred at each network device (e.g., routers/switches), and $H(dc_k^{m,j})$ is the number of network devices in the kth datacenter that transfers the data between workflow tasks m and j.

Multiple datacenters. The VMs of a_1 and a_2 can be also allocated at different datacenters. For instance, if the vm_1 which is used to host a_1 is deployed on dc_1, and vm_2 which is used to host a_2 is deployed on dc_2; then, the time for transferring P size of data from a_1 to a_2 is acquired as $T_t(a_m, a_j) = \frac{P}{min(tp_{vm_i^m}, tp_{vm_x^j})} + P \times \varphi \times H(dc_k^m, dc_q^j)$, where φ is the average latency incurred at each network device and $H(dc_k^m, dc_q^j)$ is the number of network devices between datacenters dc_k and dc_q, that are involved in the transmission process, including the network devices of each datacenter.

The makespan of each workflow depends on the performance of its allocated VMs and network. Thus, the makespan of executing a workflow λ is computed using Equation 5.2 where $\mathcal{A} \subset \lambda$ is a set of the activities (e.,g, executions, transmissions) that belong to λ.

$$Makespan(a_m, vm_i) = T_l(vm_i^m) + T_e(vm_i^m) + T_t(vm_i^m, vm_x^j) \quad m, j \in \mathcal{A}, m \neq j, i \neq x \tag{5.2}$$

5.3.3 Energy model

A data-driven workflow application mainly consumes energy when performing two main operations: a) data processing or computation and b) network communication. For computation, in general, the power consumption of a server/host varies as a function of its utilization level. If a host is idle, the power saving mechanism lowers the

frequency of the CPU, and therefore only a small proportion (α) of peak power is utilized. If ρ is peak power consumption and u is the utilization of resources, then the power consumption of a host in a single datacenter can be computed using Equation 5.3.

$$P_{host}^{comp} = \alpha \times \rho + (1 - \alpha) \times \rho \times u \tag{5.3}$$

A host may run several VMs at a given time. Each VM will consume its host's energy according to its usage of resources. Even though a host may have several resources such as CPU cores, disks, memories, and other elements, we assume that u_{vm_i} indicates aggregate resources utilized by each VM i hosted on a given host. As such, the power consumption of a host is formalized using Equation 5.4.

$$\mathcal{P}_{host}^{comp}(a_m, vm_i) = \alpha \times \rho + (1 - \alpha) \times \rho \times u(vm_i^m) \tag{5.4}$$

For modeling network energy consumption, we assume that network devices consume energy in two ways: one while running network devices' operating systems and associated operations while the other when sending data through active network ports/links. Let $i, m \rightarrow x, j$ denote the data transmission from the ith VM that hosts a workflow task m to the xth VM that hosts another workflow j. For computing end-to-end network energy consumption for every given data among workflow tasks, Equation 5.5 is used where $\mathcal{P}_{switch}$ is the required energy for operating network devices, $\mathcal{P}_{port}$ represents network ports that are involved in the transmission process, and $s \in S$ denotes a network device. Based on Equation 5.4 and 5.5, the total energy consumption of $a_m \in \mathcal{A}$ can be formalized using Equation 5.6.

$$\mathcal{P}_{network}^{comm}(a_{i,m\rightarrow x,j}) = \sum_{s \in S} \mathcal{P}_{switch}^{i,m\rightarrow x,j}(s) + \mathcal{P}_{port}^{i,m\rightarrow x,j}(s) \qquad m, j \in \mathcal{A}; i, x \in \mathcal{VM} \tag{5.5}$$

$$TEnergy(a_m, vm_i) = \mathcal{P}^{comp}_{host}(a_m, vm_i) + \mathcal{P}^{comm}_{network}(a_{i,m\rightarrow x,j}) \tag{5.6}$$

5.3.4 Electricity cost

The electricity cost is computed according to data processing and communication. We compute the electricity cost of data processing by multiplying the local electricity price of a given datacenter with energy consumed by a corresponding VM i that execute a workflow task m, as shown in Equation 5.7, where $Eprice(vm^m_i)$ represents the electricity price of vm^m_i.

$$\mathcal{E}^{comp}_{host}(a_m, vm_i) = \mathcal{P}^{comp}_{host}(a_m, vm_i) \times Eprice(vm^m_i) \tag{5.7}$$

Regarding the electricity cost occurred as a result of data exchange, we assume that the electricity price is a constant value Ω for each network device. As such, the network electricity cost is acquired using Equation 5.8. Based on Equation 5.7 and 5.8, the total electricity cost for $a_m \in \mathcal{A}$ is formalized in Equation 5.9.

$$\begin{aligned} \mathcal{E}^{comm}_{network}(a_{m\rightarrow j}, vm_{i\rightarrow x}) = \mathcal{P}^{comm}_{network}(a_{m\rightarrow j}, vm_{i\rightarrow x}) \times \Omega \\ m, j \in \mathcal{A}; i, x \in \mathcal{VM} \end{aligned} \tag{5.8}$$

$$TEle(a_m, vm_i) = \mathcal{E}^{comp}_{host}(a_m, vm_i) + \mathcal{E}^{comm}_{network}(a_{m\rightarrow j}, vm_{i\rightarrow x}) \tag{5.9}$$

5.4 Proposed energy-aware GA algorithm (GreenGA)

The aim of this chapter is to find an optimized solution that maximizes renewable energy usage and minimizes the electricity cost under users' deadline constraints. This can be considered as a dual objective optimization problem. Given the complexity of the workflow scheduling problem with multiple objective functions and constraints

(which is defined as an NP-hard problem), it is not possible to find an optimal solution in a polynomial time. Thus, this chapter adapts a Genetic Algorithm (GA), a well-known evolutionary algorithm capable of finding a near-optimal solution for data-driven workflow deployments. Previously, the GA has been applied for optimizing the makespan of data-driven applications [116, 117]; however, its applicability and performance have not been evaluated to optimize different factors (e.g., deadline, cost, energy consumption, and carbon footprints) with the embracing of SD-WAN capabilities.

In our GA approach, we first converted this multi-objective problem into a single objective optimization problem by multiplying each objective. The resultant problem is formulated in Equation 5.10 where $\mathcal{N}$ indicates the non-green energy consumption, $\mathcal{C}$ represents the total monetary cost for running a given workflow λ, and $a_{m,s} \in \mathcal{O}$ represents one deployment solution for task a_m. $\sigma(a_{m,s})$ is a function that calculates the proportion of renewable energy consumption of the deployment solution $a_{m,s}$. Finally, *deadline* and *budget*, which are given by users, denote the hard constraints for running a workflow λ.

$$
\begin{aligned}
\textbf{minimize} \quad & \sum_{a_{m,s} \in \mathcal{O}} (\mathcal{N}(a_{m,s}) \times \mathcal{C}(a_{m,s})) \\
s.t. \quad & \mathcal{N}(a_{m,s}) = (1 - \sigma(a_{m,s})) \times TEnergy(a_{m,s}, vm_i) \\
& \mathcal{C}(a_{m,s}) = TEle(a_{m,s}, vm_i) \\
& \sum_{a_{m,s} \in \mathcal{O}} Makespan(a_{m,s}, vm_i) \leq deadline \\
& \sum_{a_{m,s} \in \mathcal{O}} UCost(a_{m,s}, vm_i) \leq budget \\
& \forall vm_i \in VM
\end{aligned}
\tag{5.10}
$$

5.4.1 Algorithm details

The GA aims to search its solution space and find the best values for the given objective function (minimization of non-green energy usage and energy cost). To this end, the objective function in Equation 5.10 is encoded into a deployment solution $\mathcal{O}$ using a vector $[a^1_{z,c}; a^2_{b,y}$ :::: $a^q_{w,e}]$, where $a^k_{m,s}$ denotes a deployment solution s for an activity

a_m (e.g., processing) deployed on a cloud datacenter dc_k. Therefore, the vector can be used to compute the values of the objective function based on the three proposed models (cost, energy, and performance), in addition to the constraints (deadline and budget). After encoding, we can perform the adaptive GA to compute near-optimal solutions through the following four phases:

1. *Initializing population and candidate list generation:* Initially, we randomly generate the population, which is coded according to the objective function. We select cloud datacenters from the "Candidate List" that meet the constraints (e.g., deadline) of a given workflow activity to reduce the possibility of generating an infeasible solution.

2. *Selection:* The selection process is based on two methods: fitness function and diversity mechanism. The fitness function is represented in a numeric format, which is used to select superior solutions. The fitness function is the same as the objective function, which aims to minimize the non-green energy usage and energy cost as well as meeting the constraints. To prevent superior individuals from being destroyed in the crossover and mutation (discussed in the next phase), the elitism method [124] given in Algorithm 2 is used. So, if a solution is tagged as an elitist, it should be a part of the new population's generation process. This method can ensure that the proposed GA algorithm is computationally efficient as it avoids the re-discovery process of good results, which have been already obtained in previous generations. The diversity mechanism performed in the crossover and mutation phase is a crucial step to consider, as it impacts the future steps of crossover and mutation. The low diversity of a population usually indicates a local extreme, which impacts the search for optimal solutions.

3. *Crossover and mutation:* A crossover aims to exchange some parts of two chromosomes to generate two new chromosomes. The proposed GA algorithm uses one-point crossover [125]. A mutation is used to enhance the search range by avoiding local minima; therefore, the algorithm consists of a method that randomly selects a small proportion of given chromosomes in the current generation and changes them to new feasible chromosomes for the next generation (as de-

Algorithm 2 Elitist Prevention

Require:
elist: elitist list
s: elitist size
pop: all individuals
$\mathcal{T}$: task list
$\mathcal{C}$: cloud list
1: **if** *elist* is empty **then**
2: ▷ ASCsort sorts the *pop* as ascending order
3: *pop* ← ASCsort (*pop*)
4: ▷ copy the first s number of solutions to *elist*
5: *elist* ← from *pop*[0] to *pop*[$s-1$]
6: **end if**
7: **for** t in $\mathcal{T}$ **do**
8: **for** c in $\mathcal{C}$ **do**
9: *pop* ← combine(*elist*, *pop*)
10: *pop* ← ASCsort (*pop*)
11: ▷ delete s numbers of *pop* in tail
12: *elist* ← from *pop*[0] to *pop*[$s-1$]
13: **end for**
14: **end for**

scribed in [126]). The initial mutation rate is set to 0.015, which is a very small value. Nevertheless, the proposed algorithm can dynamically adjust the mutation rate, detailed in the following phase.

4. *Diversity mechanism:* Algorithm 3 illustrates the procedure used to dynamically adjust the diversity of a chromosome based on mutation rates. Firstly, the algorithm computes the density d of the population by comparing unique chromosomes (sr) with the total number of population ($size$). Next, it increases the mutation rate if d is less than a predefined threshold. However, if the mutation rate is higher than its maximum rate, it will be decreased. The increasing and decreasing step is computed as $\frac{1.75}{\kappa \times |pop|}$ as recommended by Smith et al. in [127]. κ denotes the length of chromosomes.

5.4.1.1 Termination method

If the number of iterations is ∞, the proposed GA algorithm can provide an optimal solution. However, as the computation resource is limited, the proposed GA algorithm should be terminated at some points. Therefore, the algorithm would terminate itself when there is no further improvement for a given solution in a fixed number of interactions R.

Algorithm 3 Diversity mechanism

Require:
pop: all individuals
size: size of *pop*
threshold: threshold of diversity
rate: current mutation rate
Max: maximum mutation rate

1: ▷ function *removeDup* removes the duplication
2: $rpop \leftarrow removeDup(pop)$
3: $sr \leftarrow |rpop|$
4: $d \leftarrow 1 - \frac{sr}{size}$
5: **if** $d < threshold$ **then**
6: increase *rate*
7: **else if** $rate > Max$ **then**
8: decrease *rate*
9: **end if**

5.4.1.2 Time Complexity

The proposed method is split into four operators: selection, crossover, mutation, and diversity mechanism. The time complexity of the selection phase is $O(|P| \times |G| \times |\mathcal{T}|)$, where P is the size of the population, G is the number of generations, and $\mathcal{T}$ represents the total number of tasks of a given workflow. For the crossover phase, each selected individual needs to be processed in every generation, so the crossover's complexity is $O(|P| \times |G|)$. Although the mutation is not applied to each selected individual in each generation, its rate must be computed in the diversity mechanism phase, which requires to sort the solutions. Thus, the time complexity of the mutation and diversity mechanism together is $O(|P| \times |G|)$. As a result, the proposed method's overall time complexity is $O(|P| \times |G| \times |\mathcal{T}|)$.

5.5 Performance evaluation

5.5.1 Experimental setup

To evaluate the proposed GA algorithm [128], we use our proposed simulator "IoTSim-SDWAN," as discussed in the previous chapter. IoTSim-SDWAN is capable of simulating multiple cloud datacenters interconnected via traditional WAN or SD-WAN environments. IoTSim-SDWAN is developed on top of CloudSim [129], which is one of the most used cloud-based simulators worldwide. CloudSim was evaluated and compared with several real-world test-beds in many scenarios, including deploying scientific and data-driven workflows on multiple clouds [117, 130].

Figure 5.2: Geo-distributed cloud datacenters (D* denotes a datacenter number)

VM type	Small	Medium	Large
Price	10.5 ($/h)	12.8 ($/h)	30 ($/h)
CPU	70 (MIPS)	80 (MIPS)	100 (MIPS)
Storage size	1000 (GB)	2000 (GB)	4000 (GB)
RAM	512 (MB)	2000 (MB)	6000 (MB)
Energy consumption	4.5 (kWh)	6.5 (kWh)	10.5 (kWh)

Table 5.1: The configuration of VMs

5.5.1.1 Cloud provider configurations

- *Datacenter location and proportion of green energy:* The chapter assumes that there are six cloud datacenters, which are allocated in different countries, as shown in Figure 5.2. The figure illustrates how the geo-distributed datacenters are interconnected with each other via an SD-WAN network topology, where the SD-WAN bandwidth is set to 1 GB for all links. The weight of each SD-WAN route represents the number of network devices that exist between two respective datacenters. Data have to traverse through these network devices to reach their destinations. The green energy usage of each datacenter is given based on an ascending datacenter order and as follows: 0.895, 0.895, 0.934, 0.932, 0.622, and 0.071.

- *VM configuration:* The chapter assumes that each datacenter has three types of

Workflow	Medium	Large	Very large
CyberShake	30	100	1000
Montage	25	100	1000
LIGO	30	100	1000
Epigenomics	24	100	995

Table 5.2: Number of tasks of each workflow at each scale.

VMs: small, medium, and large. Table 5.1 shows the configuration of each VM type. For example, the small VM costs 10.5 US Dollars per hour, and its CPU, storage size, and RAM are 70 MIPS, 1000 GB, and 512 MB, respectively. The MIPS describes the CPU powers in Millions of Instructions Per Second. The small VM is also assumed to consume a total of 4.5 kilowatts of electric energy per hour (kWh).

- *Electricity Prices:* Market electricity prices vary based on several variables: country, hour, and day. To simulate each datacenter's market electricity cost, we used the energy markets' prices in the United Kingdom observed over one week.[2]

5.5.1.2 User Configuration

- *Workflow generation:* To evaluate the proposed GA algorithm, four data-driven workflow applications are considered: Epigenomics (bioinformatics), CyberShake (earthquake risk characterization), LIGO (detection of gravitational waves), and Montage (generation of sky mosaics). The XML description files of the workflows are available via the Pegasus project.[3] Table 5.2 shows the number of tasks of each workflow application. Notably, our evaluations only consider input and output data sizes, VM's processing for each given task, and SD-WAN network traffic among geo-distributed datacenters. The internal network traffic of workflows between tasks in a given datacenter is not considered.

- *Deadline generation:* In this chapter, the deadline is a hard constraint defined as the mean of *fastest solution* and *slowest solution*. *Fastest solution* is the optimal deployment solution, which means to deploy a given workflow over the most

[2]http://www.nordpoolspot.com, accessed 01-06-2015

[3]μ`https://confluence.pegasus.isi.edu/display/pegasus/WorkflowHub`

powerful VMs in the same datacenter, while the *slowest solution* is to deploy the workflow over the least powerful VMs across different datacenters.

5.5.2 Experimental results

The performance of the proposed GA algorithm is evaluated based on five objectives: electricity cost, energy consumption, deadline, the usage of renewable energy, and the level of energy efficiency using SD-WAN as compared to a classical WAN infrastructure. To this end, we compare the solutions generated by our proposed algorithm with the best and worst case of each objective. Moreover, we evaluate our adaptive algorithm's performance in two ways: 1) to optimize only one objective and keep others within predefined constraints (EleCostGA); 2) to optimize more than one objective by mapping these objectives into a weighted linear function while ensuring other objectives within the predefined constraints (GreenGA).

5.5.2.1 Electricity cost

We develop two versions of GA-based algorithms, namely EleCostGA and GreenGA to optimize data-driven workflow deployments across multiple datacenters. EleCostGA only aims to minimize electricity costs while meeting other constraints (makespan, energy consumption, and user budgets). On the other hand, GreenGA aims to minimize both electricity cost and consumption of non-renewable energy and meet the same constraints as in EleCostGA (makespan, energy consumption, and user budgets).

Figures 5.3, 5.4, and 5.5 show the results of applying both algorithms to different types of workflows. The Lower_bound represents the lowest electricity cost obtained without considering any constraints, where Y-axis is the ratio of the results generated by the proposed algorithms with the Lower_bound (e.g., $\frac{EleCostGA}{Lower\_bound}$ and $\frac{GreenGA}{Lower\_bound}$). The results illustrate that the cloud providers have to spend more when they optimize the proportion of renewable energy usage. However, with the increasing size of the workflows, the differences in electricity cost for GreenGA and ElecCostGA become smaller. Figure 5.6 shows the relation between the number of generations of the propped GA algorithm and electricity cost saving. We first set the result of 10 generations as the

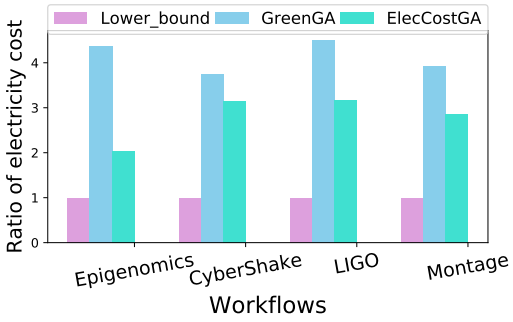


Figure 5.3: Electricity cost for medium size workflow

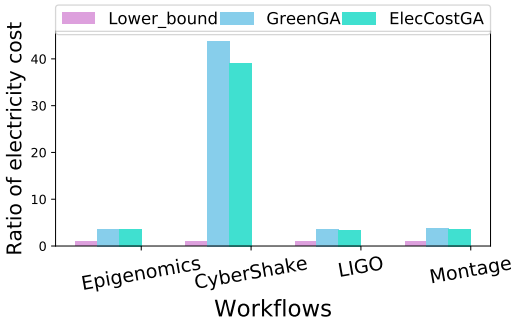


Figure 5.4: Electricity cost for large size workflow

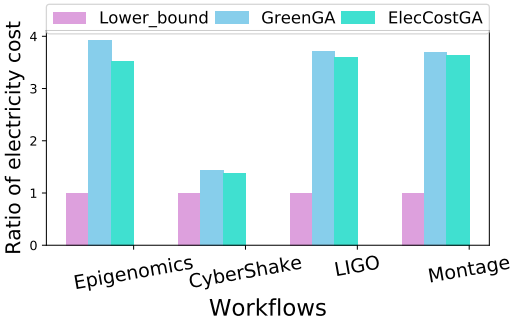


Figure 5.5: Electricity cost for very large size workflow

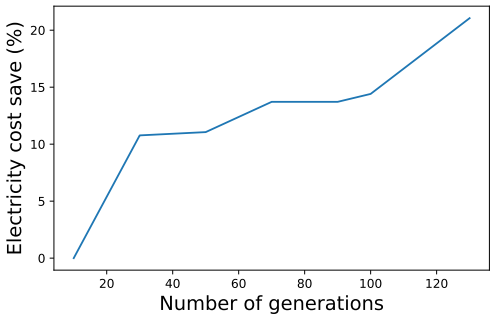


Figure 5.6: Electricity cost vs number of generation

baseline and then compute how many percentages can be saved when the number of generations is increased.

5.5.2.2 Energy efficiency

In this chapter, we do not aim to minimize the energy consumption for executing the workflows of data-driven applications across a set of datacenters. Nevertheless, our proposed method allows cloud providers to set a constraint for energy consumption. To this end, we first provide the Lower_bound and Upper_bound of energy consumption for executing the workflows over available datacenters.

Lower_bound and Upper_bound. The Lower_bound of energy consumption is computed by selecting the most energy efficient VMs inside the same datacenter to execute a given workflow. So, the most energy efficient VM (vm_i) for a task a_m is determined using: $\arg\min_{vm_i \in VM} Makespan(a_m, vm_i) * \mathcal{P}^{comp}_{host}(a_m, vm_i)$. On the other hand, the Upper_bound aims to select VMs that consume the most energy among datacenters.

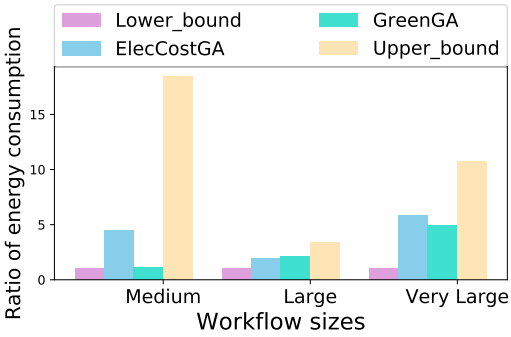


Figure 5.7: Energy consumption for Epigenomics

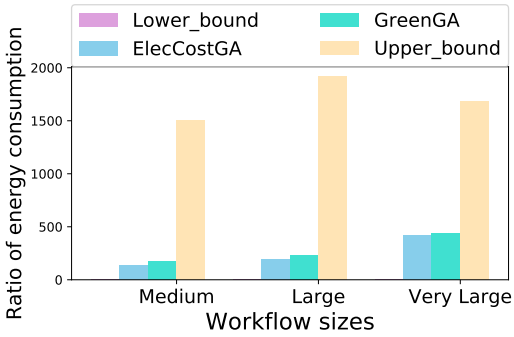


Figure 5.8: Energy consumption for CyberShake

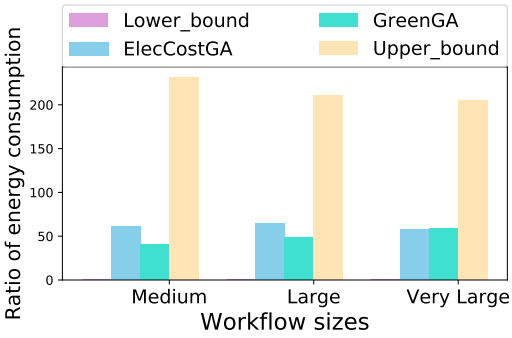


Figure 5.9: Energy consumption for Montage

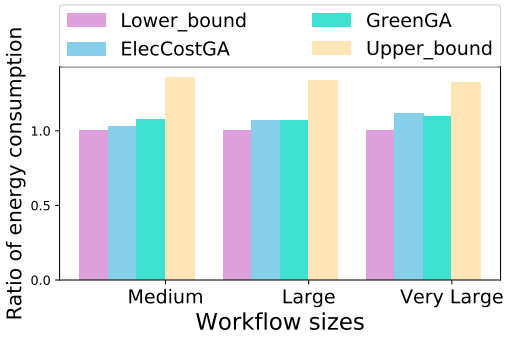


Figure 5.10: Energy consumption for LIGO

Figures 5.7, 5.8, 5.9, and 5.10 indicate the energy consumption of executing the given workflows over different datacenters. The Y-axis indicates the ratio of the energy consumption of different solutions (GreenGA, EleCostGA, and Upper_bound) with the Lower_bound's energy consumption. Although both GreenGA and EleCostGA are not designed to minimize energy consumption in general, they can guarantee that the energy consumption will meet the predefined constraints while at the same time minimize the electricity cost.

5.5.2.3 Green energy efficiency

This subsection shows the proportion of renewable energy usage of each solution generated by GreenGA and EleCostGA. Figures 5.11, 5.12, and 5.13 show that the GreenGA's solutions use more green energy than those generated by EleCostGA. However, the difference reduces as the size of workflows increases. The larger size of workflows correspond to more deployment solutions; which sometimes leads a local

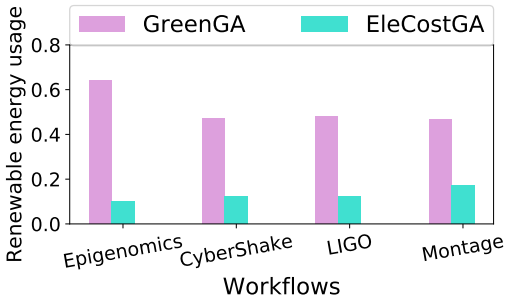


Figure 5.11: The proportion of the usage of renewable energy for medium size workflow

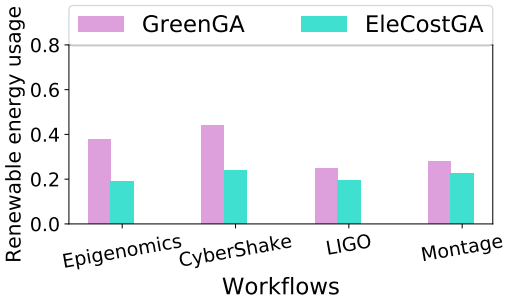


Figure 5.12: The proportion of the usage of renewable energy for large size workflow

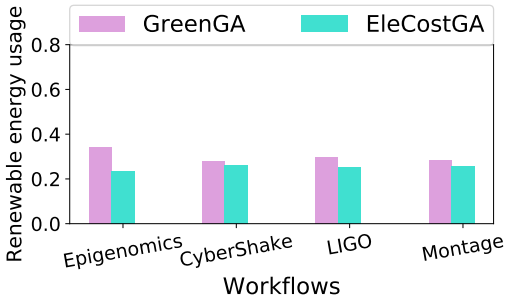


Figure 5.13: The proportion of the usage of renewable energy for very large size workflow

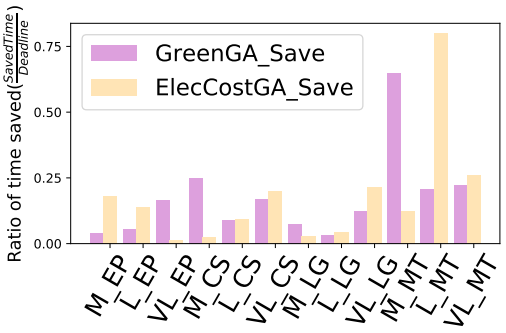


Figure 5.14: The presentation of time save comparing with deadline

optimum. The local optimum has a very high probability of causing the proposed GA algorithms to terminate when meeting the predefined termination conditions, as described in Subsection 5.4.1.1.

5.5.2.4 Performance (makespan)

Deadline is a hard constraint, which means that the execution time of each submitted workflow must be equal to or less than a specified deadline. Figure 5.14 shows the time saving of the generated solutions. The figure compares the saving time with given deadlines, where the Y-axis represents the ratio of saving time and the given deadlines ($\frac{savedTime}{deadLine}$). The X-axis denotes to the type of workflow; for example, "M_EP", "L_EP" and "VL_EP" represent the medium, large, and very large size of Epigenomics workflows, respectively. The results in the figure show that all of the generated solutions meet the given deadlines.

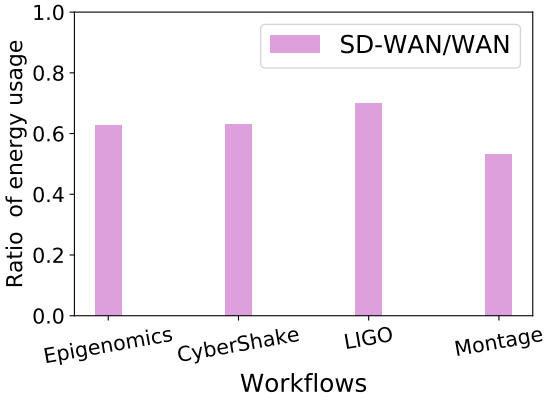


Figure 5.15: Energy consumption in a small-scale SDWAN/WAN - medium size workflows

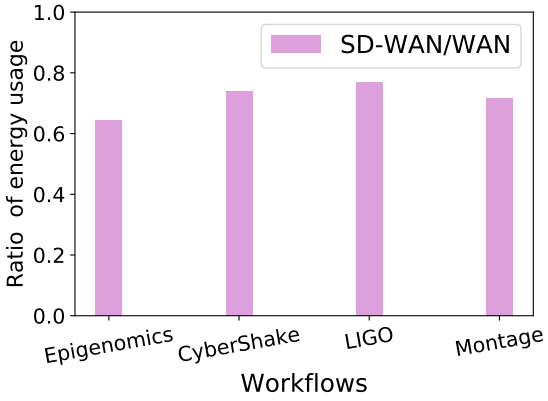


Figure 5.16: Energy consumption in a small-scale SDWAN/WAN - large size workflows

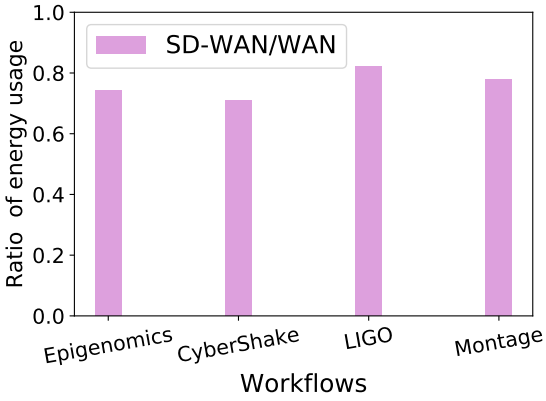


Figure 5.17: Energy consumption in a small-scale SDWAN/WAN - very large size workflows

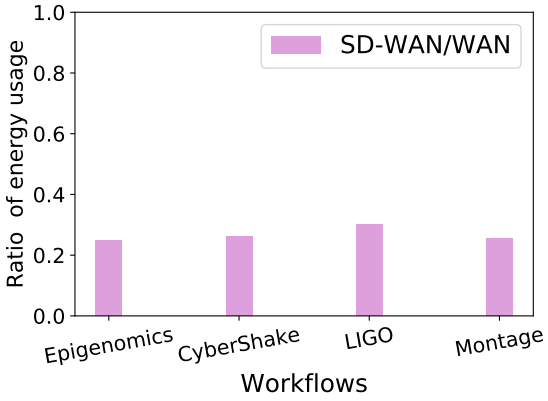


Figure 5.18: Energy consumption in a large-scale SDWAN/WAN - medium size workflows

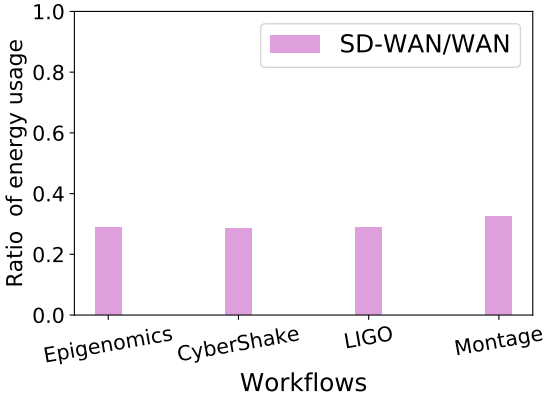


Figure 5.19: Energy consumption in a large-scale SDWAN/WAN - large size workflows

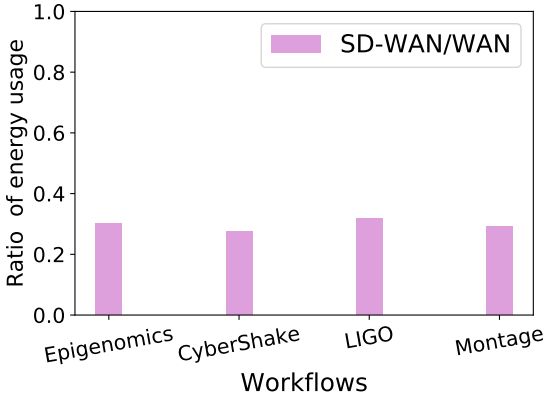


Figure 5.20: Energy consumption in a large-scale SDWAN/WAN - very large size workflows

5.6 Energy efficiency in SD-WAN

To evaluate SD-WAN's energy efficiency and flexibility when transferring the data of data-driven applications, we also use IoTSim-SDWAN [128]. As discussed in the previous chapter, IoTSim-SDWAN provides the facilities to evaluate networks' energy consumption in both traditional WAN and SD-WAN environments.

Experiment configuration. We consider two types of network topologies: *small scale SD-WAN/WAN* and *large scale SD-WAN/WAN*. The number of network devices in the large scale SD-WAN/WAN are twice as in the small scale SD-WAN/WAN. Regarding the datacenters and workflows, we keep their configurations similar to the experiments performed in the previous section. In the traditional WAN environment, we identify a set of shortest paths and then randomly select one of the paths to be the prime paths for data transmission. In the SD-WAN environment, we use our proposed SPMB algorithm, discussed in the previous chapter, to transfer data among distributed datacenters via SD-WAN networks (refer to Subsection 4.4.3).

We report the experimental results as the ratio of the energy consumption of SD-WAN-based solution and WAN-based solution ($SD-WAN/WAN$). Figures 5.15, 5.16, and 5.17 show that the SD-WAN-enabled environment consumes less energy than the WAN environment for transferring data across multiple datacenters in the small-scale network topology. The SD-WAN solution can save network energy up to 32.5% as compared to the WAN solution. However, the advantage reduces when the size of the workflow increases. For example, from medium-sized workflows to very large-sized workflows, the energy-saving of SD-WAN compared to WAN is reduced from 38.6% to 27.4%. This is because the SD-WAN has fewer paths to consider when transferring data in small-scale networks.

When the network topology becomes bigger and more complicated, the advantage of the SD-WAN becomes more significant. Compared to the WAN solution, the SD-WAN solution saves network energy up to 73.8%, as shown in Figures 5.18, 5.19, and 5.20. Similar to the small-scale case, this advantage degrades when the size of the workflows increase. However, the degradation is slight; for example, from medium-sized workflows to very large-sized workflows, the energy saving is reduced from 77%

Table 5.3: Compare with other related work

Feature/ Research work	[120]	[121]	[131]	[118]	[117]	[116]	[132]	[133]	Proposed work
Workflow application	×	×	×	×	✓	✓	×	×	✓
SD-WAN aware	×	×	×	×	×	×	×	×	✓
Multiple clouds	×	×	✓	×	✓	✓	✓	✓	✓
Green energy	✓	×	×	×	×	×	✓	✓	✓
Energy minimization	×	✓	✓	×	×	×	✓	✓	✓
Cost minimization	✓	✓	✓	×	✓	✓	✓	✓	✓

to 70.75%.

5.7 Related work

5.7.1 Cost and performance-based tasks scheduling

To improve the performance of complex workflow-based applications, Mao et al. [134] introduced an auto-scaling method to allocate workflow tasks to a set of cloud VMs according to user deadline constraints. Malawski et al. [135] considered the constraint of monetary cost for scheduling large-scale, workflow-based applications in cloud datacenters. An algorithm was proposed to make trade-off decisions between makespan and financial cost. Moreover, a new algorithm was proposed by Calheiros et al. [136] to accelerate the execution time of complex workflow applications in cloud datacenters. The algorithm takes advantage of idle cloud VMs, in addition to meeting a number of constraints (e.g., deadline, user budget).

In multi-distributed cloud datacenters, Yuan et al. [131] proposed an effective algorithm for scheduling tasks across private and public datacenters, considering the objective of minimizing cost and overall delays. PANDA [137] was also developed to schedule workflow-based applications across private and public datacenters, aiming to find the best trade-off decisions between performance and cost. Fard et al. [138] proposed a solution that considers the trade-off between monetary cost and completion time using a Pareto-optimal based algorithm. Jrad et al. [139] introduced a static method to optimize the deployment of workflow-based applications on multiple clouds,

considering security, makespan, and monetary cost.

5.7.2 Green scheduling

Various existing proposals suggested different solutions with respect to green scheduling in cloud computing [132, 133]. Catena et al. [140] proposed a predictive energy saving algorithm based on online scheduling to reduce the energy consumption of distributed web search engines. Y.Li et al.[141] proposed an energy model across edge and cloud environments. The author aimed to estimate the energy consumption based on the number of IoT devices and the desired application QoS. There are many studies and solutions [119, 142–144] proposed to improve the performance of cloud datacenters in terms of minimizing electricity usage and carbon footprint. For instance, Aksanli et al. [142] proposed a database scheduling strategy which predicts the green energy availability in order to reduce the number of rescheduled tasks due to green energy unavailability. Goiri et al. [143] also proposed a prediction mechanism that aims to maximize green energy consumption while minimizing the use of traditional energy sources. The author focused on the workflows of big data applications (MapReduce). Moreover, Deng et al. [144] proposed an online algorithm to ensure energy reliability and minimize the operational cost of cloud datacenters by using multiple energy resources in a complementary manner.

Most of the above studies focus on single datacenters. In multi-distributed cloud datacenters, Garg et al. [145] proposed a cloud-based framework that aims to minimize carbon footprints introduced by cloud datacenters. The author's framework depends on the usage proportion of green energy to select the most green-aware cloud provider, along with the consideration of user's QoS preferences. Kaushik et al. [146] proposed an energy-saving solution for cloud datacenters by using data classification, power characteristics, and idleness. The author's solution aims to divide cloud storage structures into different cloud datacenters, which leads to the minimization of cloud providers' energy usage and operational costs. Kiani et al.[147] proposed a solution that increases green energy utilization by distributing a given workload into several cloud datacenters based on residual green energy rate. To find an optimal green-energy datacenter, the author applied the technique of brute-force optimization. Giacobbe et

al. [123] introduced an approach to minimize clouds' carbon dioxide emissions by migrating VMs and their workloads among distributed federated clouds. The study uses a percentage quota of renewable energy to decide its optimal placement of VMs and workloads. Yuan et al. [120] proposed a time-aware task scheduling algorithm based on a hybrid chaotic particle swarm optimization. The author's proposed solution aims to meet user constraints (e.g., task deadlines) while at the same time maximizing the proportion of green energy and cloud providers' profits.

In summary, to the best of our knowledge, our proposed GA-based scheduling method is the first work that focuses on maximizing the usage proportion of green energy and minimizing electricity cost for data-driven applications executed in multiple SD-WAN-enabled cloud datacenters. Our work differs from the above studies by considering cross-layer optimizations in terms of application, server, and network. By leveraging an evolutionary search process (genetic algorithm), our work is capable of finding near-optimal solutions in a time-efficient manner. Table 5.3 provides a comparison between our proposed work and the state-of-the-arts.

5.8 Conclusion

This chapter considers the problem of finding a suitable deployment solution for data-driven applications in geo-distributed cloud datacneters. We argue that, in order to find the best deployment solution that minimize non-green energy and energy cost as weel as meeting other constraints (e.g., user's deadline and budget), it is necessary to embrace SD-WAN capabilities and efficient search algorithms. In this context, this chapter proposed an adaptive genetic algorithm method that utilizes multiple SD-WAN-enabled cloud datacenters not only to improve green energy usage but also keep the cost of execution to a minimum. The performance of the proposed algorithm are evaluated using real data-driven workflows with different sizes under various configurations of virtual machines. The experimental results clearly show that our proposed algorithm favors more green energy usage along with minimizing overall energy consumption as compared with other baseline algorithms.

6

IoTSim-Osmosis: A Framework for Modeling and Simulating IoT Applications over an Edge-Cloud Continuum

Contents

Summary

To support the testing and evaluation of IoT data-driven applications that rely on data processing and transmission across multiple edge-cloud datacenters, this chapter proposes IoTSim-Osmosis– a simulation framework that enables the evaluation and validation of osmotic computing applications. The detailed related work analysis given in this chapter demonstrates that IoTSim-Osmosis is the first simulation framework that enables unified modeling and simulation of complex IoT applications over heterogeneous edge-cloud environments while interconnecting via SD-WAN. The effectiveness and practicality of IoTSim-Osmosis is demonstrated using an electricity management and billing application case study. The IoTSim-Osmosis's proposed models are evaluated using various run-time QoS parameters, such as IoT battery consumption, IoT execution time, end-to-end network transmission time, and energy consumption of edge-cloud datacenters.

6.1 Introduction

Osmotic computing sets out the principles and algorithms for simplifying the deployment of applications in integrated edge-cloud environments. It unifies many distributed systems and applications so that data delivery and analytics are optimized in addition to migrating tasks from one environment to another based on performance metrics. As osmotic computing is a new paradigm, the analysis of proposed osmotic approaches, algorithms, and QoS policies are required to undertake various "what if" investigations. To fill this gap and support an easy evaluation of new osmotic solutions, this chapter proposes IoTSim-Osmosis– an SD-WAN and SDN-aware osmotic computing toolkit.

IoTSim-Osmosis supports the 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 and SDN 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 datacenter 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).

This main contribution of this chapter is to provide a novel framework that models and simulates osmotic computing environments. In particular, the key contributions of this chapters are:

- Proposing an architecture and system models of SD-WAN and SDN-aware osmotic computing along with simulating several components used to support edge-cloud heterogeneity and IoT application complexity.
- Modeling of several system management policies for cross-layer optimizations.
- Presenting a case study to evaluate the performance of IoTSim-Osmosis using an energy (electricity) management and billing application.

The rest of the chapter is organized as follows. Section 6.2 describes osmotic computing and graph-based IoT application construction. Section 6.3 describes related work by comparing IoTSim-Osmosis with state-of-art efforts. Section 6.4 discusses the modeling capabilities of IoTSim-Osmosis and Section 6.5 provides an empirical validation of our approach. Section 6.6 provides concluding comments and future work.

6.2 Overview

6.2.1 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 Figure 6.1. The four tiers are:

Tier 1 (IoT layer): This layer can consist of sensors, actuators, Radio Frequency Identification (RFID) tags, and mobile devices. The layer senses surrounding physical environment and transfers sensed data to edge or/and cloud datacenters for further analysis [148, 149]. These devices can have different designs in terms of software, hardware, architecture, data usage, energy sources, and communication protocols. Unlike

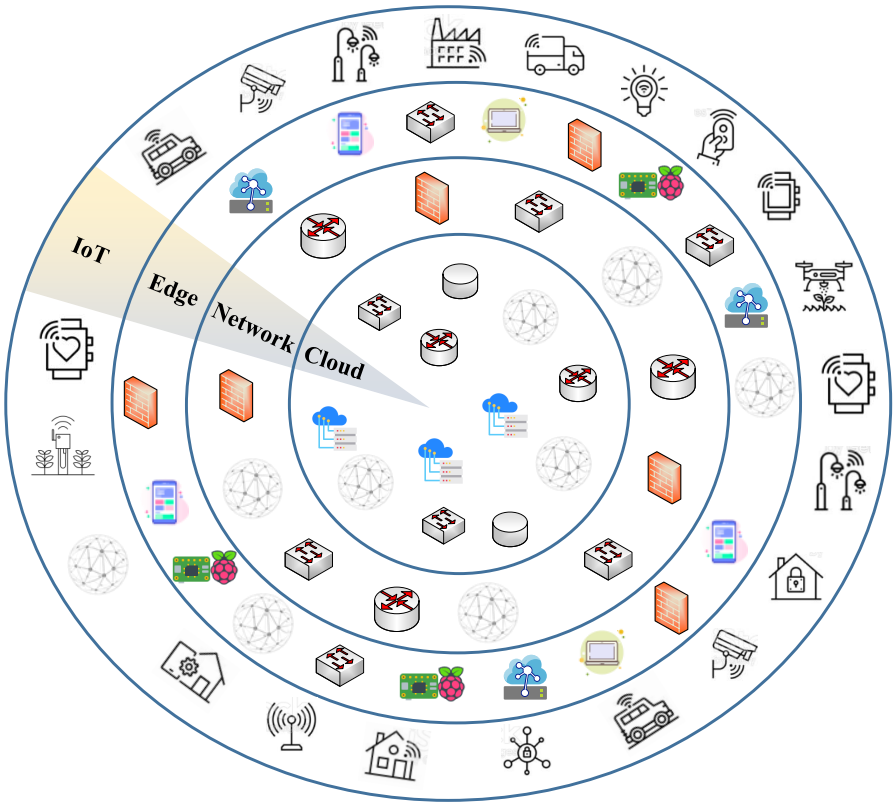


Figure 6.1: 4-tier architecture – the outer layer is composed of IoT devices generating data and transmitting these to Edge devices (second layer). The Innermost layer is comprised of a Cloud datacenter, with a data network connecting these layers

devices and networks which exist to offer physical connectivity, network-connected applications create opportunities for human-to-device connectivity [150].

Tier 2 (Edge layer): For applications with the following properties: (a) close coupling between data generators and processing environments [151], (b) network bandwidth is limited [152], and (c) data generating devices are battery operated [56], 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, favour 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

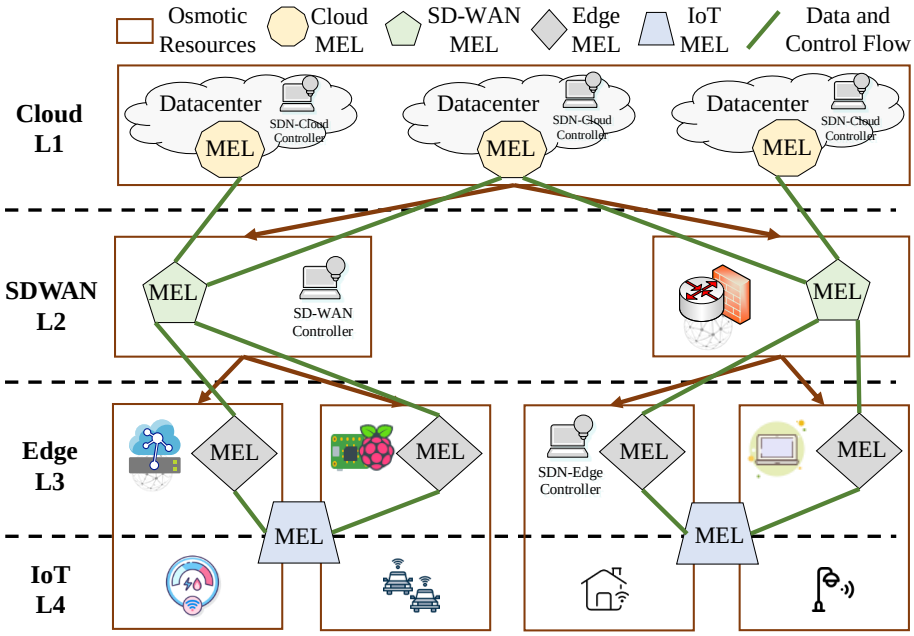


Figure 6.2: Application MEL graph

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 [53].

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 the combined IoT-edge-cloud environment.

6.2.2 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 [153]. 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 application can be represented using a graph of MELs, as shown in Figure 6.2. Modeling an application as a graph of MELs involves the following:

- **Encapsulation of multiple components:** In the context of an IoT application, sensed data needs to be processed across a number of *functions* and operations. The representation of each operation can take different forms, leading to an IoT application being specified as a graph of MELs. Each MEL can contain micro data and be realised 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 Figure 6.2.
- **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.

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

6.3.1 Cloud simulators

Multiple simulation frameworks have been proposed to model and simulate cloud computing infrastructures. The most popular one is CloudSim [1], 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 [68] extends the functionality of CloudSim to leverage traditional network infrastructures within cloud datacenters. RC2Sim [154] 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 [71] 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 hyper-visor to test different cloud brokering strategies. GreenCloud [70] 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 [155] 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 [156] 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 [157] 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).

6.3.2 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 [40]. 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 [158] 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 [159] 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 [160] 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 [161] 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++ [162] 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 the deployment

of different network-based scenarios. Castalia [163] 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 [164] 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.

6.3.3 SDN-aware network simulators and emulators

Mininet [72] 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 [2] 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 [95] is built on top of cloudSimSDN and provides models to derive different performance and network metrics of big data applications in SDN-aware cloud datacenters.

IoTSim-SDWAN [128] is a new simulation tool that provides a model of distributed SDN-aware 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-aware simulation and emulation models to evaluate different SDN performance perspectives. The focus of SDN-Sim [77] is to facilitate the deployment and testing of several SDN-aware policies, such as channel modeling, traffic shaping, and QoS demands.

6.3.4 IoT, edge, and fog simulators

In recent years, several simulators have been proposed to simulate IoT and edge environments. SimIoT [165] is another simulator which operates by modeling the trans-

Table 6.1: 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 [1]	✓								
NetworkCloudSim [68]	✓			✓					
RC2Sim [154]	✓			✓					
iCanCloud [71]	✓			✓					
GreenCloud [70]	✓			✓					
DCSim [155]	✓			✓					
Multi-RECloudSim [156]	✓								
DISSECT-CF [157]	✓								
NS-3 [40]				✓	✓				
ConesC [158]				✓					
COOJA [159]				✓	✓				
TOSSIM [160]				✓	✓				
OMNeT++ [162]				✓	✓				
Castalia [163]				✓	✓				
GreenCastalia [164]				✓	✓				
Mininet [72]			✓	✓	✓				
CloudSimSDN [2]	✓	✓		✓					
BigDataSDNSim [95]	✓	✓		✓	✓				✓
IoTSim-SDWAN [128]	✓	✓	✓	✓	✓				
SDN-Sim [77]		✓		✓	✓				✓
SimIoT [165]	✓							✓	
Edge-Fog [166]						✓	✓	✓	
iFogSim [167]	✓					✓	✓	✓	
MyiFogSim [168]	✓			✓		✓	✓	✓	
EdgeCloudSim [98]						✓	✓	✓	
IoTSim-Edge [169]					✓	✓	✓	✓	✓
Diasuite [170]					✓			✓	✓
IoTsuite [171]								✓	✓
AWS IoT Device Simulator[1]	✓			✓				✓	✓
Microsoft IoT Simulator[2]	✓			✓				✓	✓
Propoed IoTSim-Osmosis	✓	✓	✓	✓	✓	✓	✓	✓	✓

mission 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 [166] supports various energy and network models in addition to assisting with task scheduling. iFogSim [167] 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 [168] extends iFogSim and simulate network configurations, failures, and provisioning of mobile customers according to given virtual machine migration policies.

EdgeCloudSim [98] and IoTSim-Edge [169] extends the capability of CloudSim to incorporate different features of IoT and edge computing environments. While Edge-

CloudSim 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 [170] and IoTsuite [171] are the most common frameworks for managing the whole lifecycle of IoT applications. Both rely on a Siafu [172] 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-aware 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 6.1.

6.4 Design of IoTSim-Osmosis

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

[1]https://aws.amazon.com/solutions/implementations/iot-device-simulator/
[2]https://www.microsoft.com/en-us/p/iot-simulator/

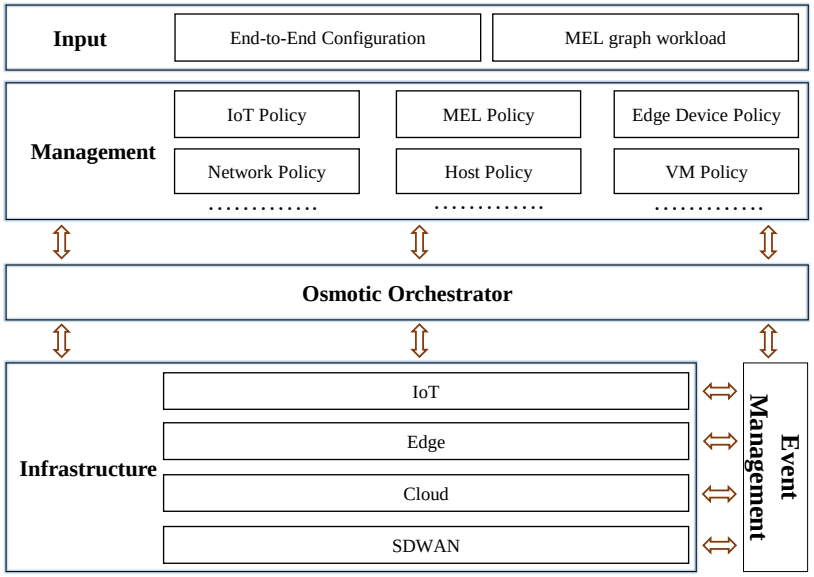


Figure 6.3: Architecture of IoTSim-Osmosis simulator

6.4.1 IoTSim-Osmosis architecture

The architecture of IoTSim-Osmosis is presented in Figure 6.3. 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, such as a network policy to instruct SDN/SD-WAN controllers with routing and traffic. As another example, VM policy is used to select a host that can deploy requested VMs. For each policy, IoTSim-Osmosis has a number of implemented algorithms that can be seamlessly used.

The *infrastructure* layer is modeled to represent four infrastructure components: IoT, edge, cloud, and SD-WAN. To provide a realistic representation of osmotic computing,

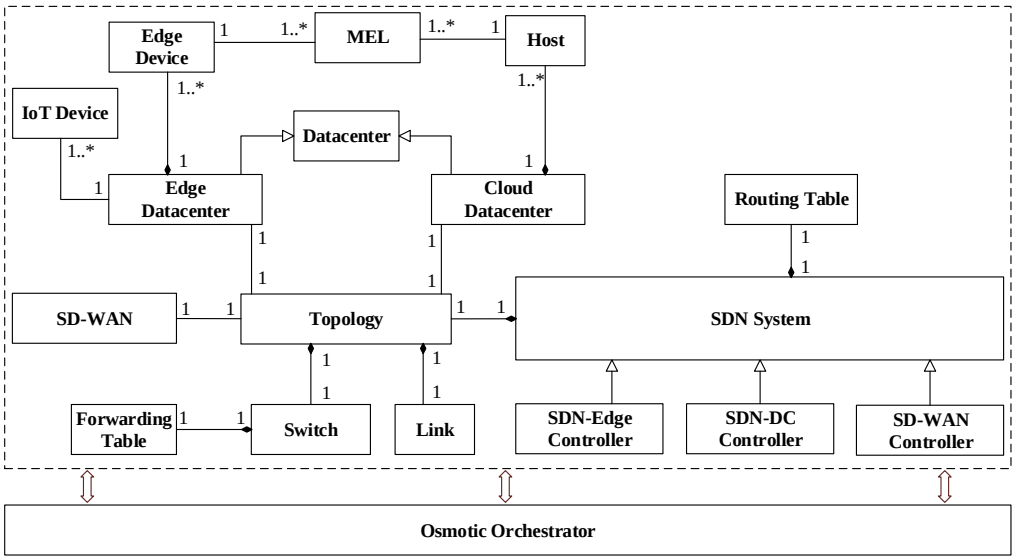


Figure 6.4: IoTSim-Osmosis system components

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. By Using the event management system, IoTSim-Osmosis is able to manage the infrastructure and network while allowing a user to apply the management policies.

6.4.2 IoTSim-Osmosis system components

An overview of IoTSim-Osmosis's system components is illustrated in 6.4. IoTSim-Osmosis has an SDN system component, which mimics the general behaviour 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 SD-WAN controller) extend the SDN system 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 SD-WAN is coupled with a topology component to describe the arrangement of the networks' nodes (e.g., edge

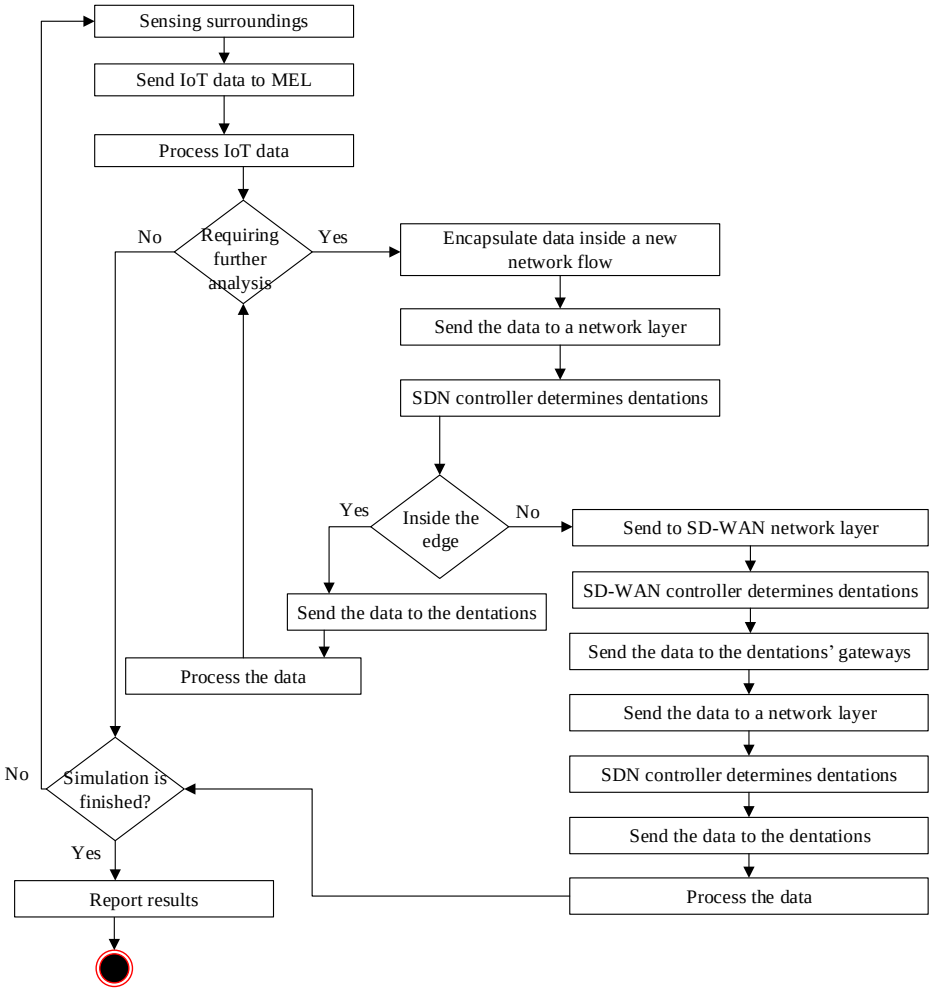


Figure 6.5: IoTSim-Osmosis overview model

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.

6.4.3 IoTSim-Osmosis model

An overview of IoTSim-Osmosis's model is shown in Figure 6.5. 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 exchanged with another MEL residing in the same edge datacenter or in another edge/cloud datacenter. The routing decision is handled by an SDN-edge controller. If the SDN-edge controller cannot determine the destined MEL, it will ask the source MEL to forward the data to its edge datacenter gateway. As an SD-WAN controller is updated with network information of all associated datacenters by the osmotic orchestrator, it easily determines a path to the destined MEL. As a result, the SD-WAN 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, Equation 6.1 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.

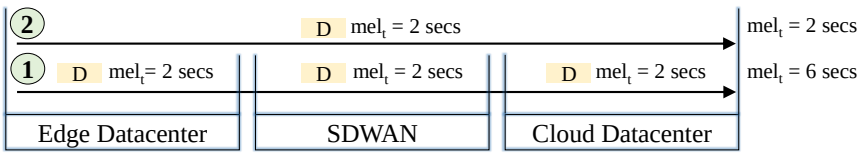


Figure 6.6: Illustration of osmotic network transmission time

$$iot_t = \frac{iot_{ds}}{min(iot_{bw}, em_{bw})} \tag{6.1}$$

More data transmissions occur when edge MELs require further analysis from 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, Figure 6.6 illustrates how the end-to-end network transmission time is computed incorrectly in step 1 and correctly in 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 seconds, which result in $mel_t = 6$ seconds. 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 $mel_t = 2$ seconds.

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 SD-WAN 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, as previously proposed in Chapter 4, the path/routing table of every controller can be 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 Equation 6.2 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 Equation 6.3 where $mel_{ds}(x)$ is the data size of the xth source MEL.

$$end_{bw}(x) = min(l(m, bw)) \quad m = 1, \forall l \in L \tag{6.2}$$

$$mel_t(x) = \frac{mel_{ds}(x)}{end_{bw}(x)} \tag{6.3}$$

To compute the processing time of each MEL, Equation 6.4 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.

$$mel_e(t) = \frac{mel_{mi}(t)}{mel_{mips}(t)} \tag{6.4}$$

Equation 6.5 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.

$$\mathcal{T} = iot_t + \sum_{\forall x \in \mathbb{X}} mel_t(x) + mel_e(x) \tag{6.5}$$

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, Equation 6.6 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.

$$\mathcal{RT}(a) = tr_s(first) - tr_f(last) \tag{6.6}$$

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 Equation 6.7 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 SD-WAN, IoTSim-Osmosis follows similar patterns as given in [2].

$$bc = s_r + t_r \tag{6.7}$$

6.5 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-aware edge, SDN-aware cloud, and SD-WAN). 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 interconnected via SD-WAN 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.

6.5.1 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

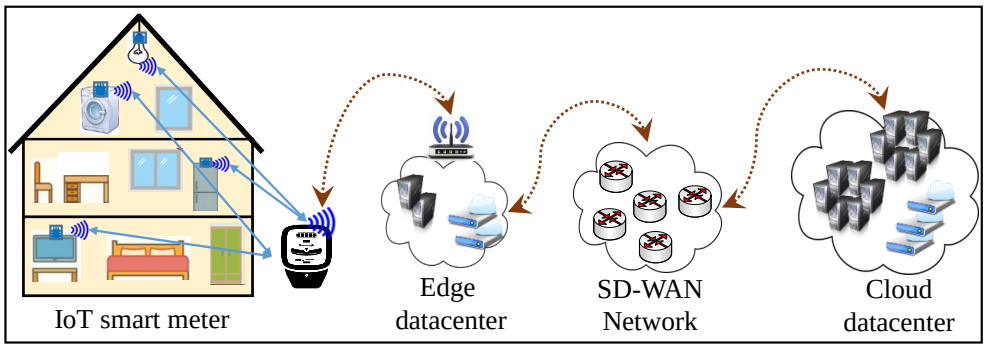


Figure 6.7: Osmotic computing example (a smart home connected to a smart city electricity meter)

Table 6.2: 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 6.3: Network configuration for the use-cases

Edge network		SD-WAN network		Cloud network	
Edge device to edge switch	100 Mbps	Edge gateway to SD-WAN router	100 Mbps	Gateway to aggregate switches	100 Mbps
edge switch to gateway	100 Mbps	Between SD-WAN routers	100 Mbps	Core switches to aggregate switches	100 Mbps
-	-	Cloud gateway to SD-WAN router	100 Mbps	Aggregate switches to edge switches	100 Mbps
-	-	-	-	Edge switches to VMs	100 Mbps

transportation for customers on demand. Each domain has various requirements (e.g., a certain level of communication delays, 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 analysis and storage. Figure 6.7 illustrates an overview of a smart home connected to a smart city meter for electricity management and billing.

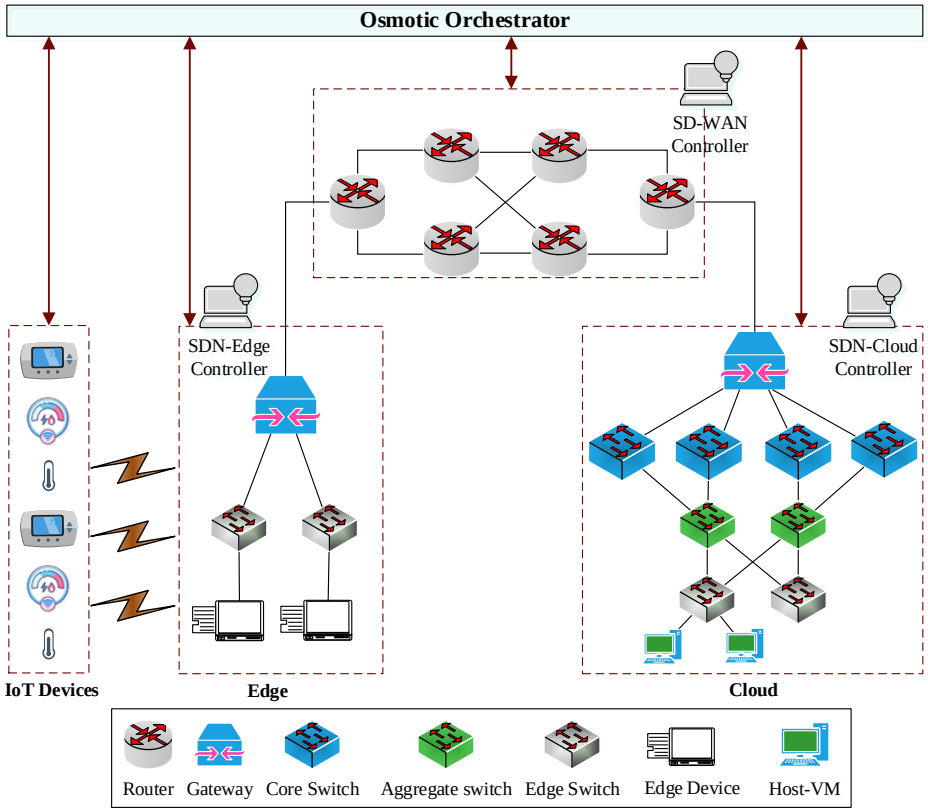


Figure 6.8: Osmosis infrastructure for case 1

Table 6.4: 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 6.5: 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

6.5.1.1 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 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

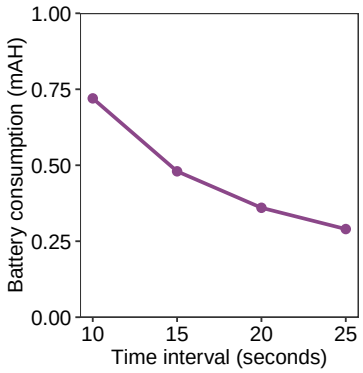
policies are used:

- 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 our previously proposed routing algorithm (SPMB), given in Chapter 4.
- The network traffic policy of IoT applications is based on a fair-share mechanism where each application obtains an equal amount of network bandwidth.

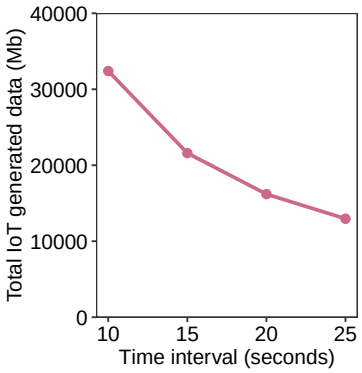
6.5.1.2 Case 1: dynamic data flow

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 Figure 6.8. Table 6.2 shows the computing configuration of edge and cloud datacenters while Table 6.3 illustrates the network configuration in the edge, cloud, and SD-WAN. Finally, Table 6.4 presents the attributes used to run each test. Table 6.5 shows the number of devices used in the case. The focus of this case is to show the effect of dynamic data generation intervals.

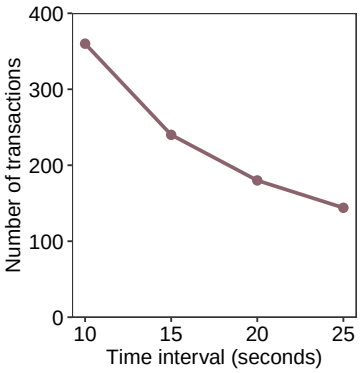
The simulation results are presented in Figure 6.9. Figure 6.9a illustrates the battery consumption of the IoT Devices. It can be observed that the lower the time interval for sending data, the higher the battery consumption. Figure 6.9b shows the total size of the generated data by IoT devices, while Figure 6.9c 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. Figure 6.9d 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. Figure 6.9e shows the total energy consumption of edge, cloud, and SD-WAN. The Figure reveals that the



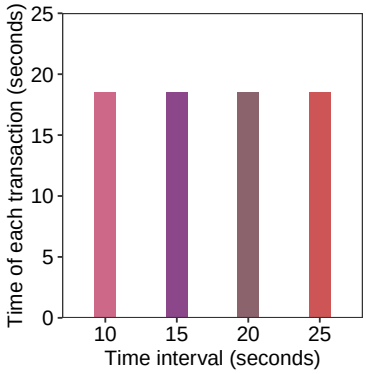
(a) Battery consumption



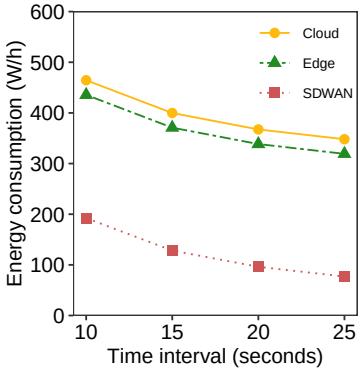
(b) Total IoT generated data



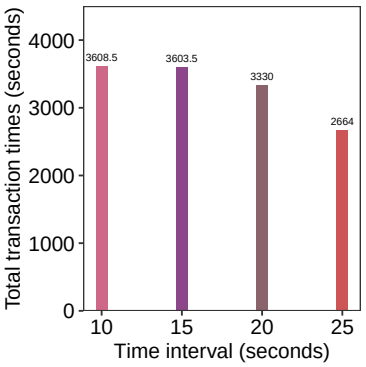
(c) Number of transactions



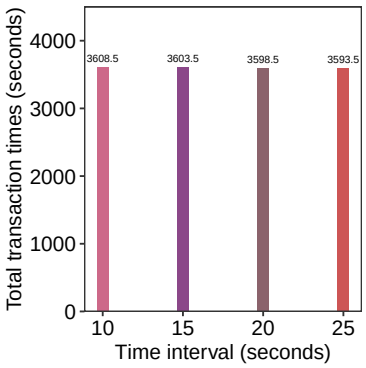
(d) Time of each IoT transaction



(e) Energy consumption of cloud, edge, and SD-WAN infrastructures



(f) Total transaction times



(g) Total running time

Figure 6.9: Simulation result for case 1

lower the time interval for IoT generating data, the higher the energy consumption. Figure 6.9f 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. Figure 6.9g illustrates the total running/simulation time. The IoT devices are set to stop generating data at 3600 seconds. It can be seen that the finishing time is not similar because the last transaction of time intervals 10,

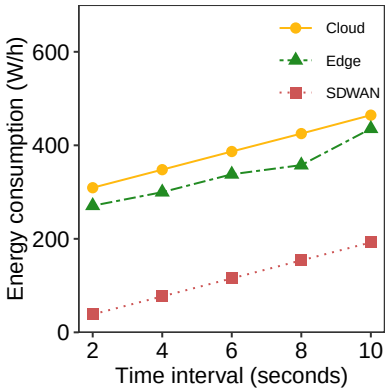


Figure 6.10: Energy consumption of cloud, edge, and SD-WAN infrastructures (use case 2)

15, and 20 requires more time to finish.

6.5.1.3 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 Table 6.2 and 6.3. This case has similar application configuration and device requirement as case 1 (see Table 6.4 and 6.5). However, the number of IoT devices varies from 2 to 10 to illustrate the impact of IoT devices on energy consumption of edge, cloud, and SD-WAN. Also, the time interval for IoT devices to generate data is set to 10. Figure 6.10 illustrates the energy consumption of edge, cloud, and SD-WAN. It is apparent that the increase in the number of IoT devices requires more energy for the edge, cloud, and SD-WAN. The battery consumption of IoT devices is neglected because data generation is static, which results in similar battery consumption for all the IoT devices.

6.6 Conclusions

The proposed simulation framework IoTSim-Osmosis provides seamless mechanisms for analyzing and validating new solutions in osmotic computing environments. In particular, IoTSim-Osmosis models 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 consumption, execution time, and energy consumption. The experimental results and related work analysis demonstrate the useful and unique simulation capabilities of IoTSim-Osmosis.

Software availability: The IoTSim-Osmosis's software with the source code can be downloaded from https://github.com/kalwasel/IoTSim-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 [128] and IoTSim-Edge [169]).

7

Conclusion

Contents

Summary

The chapter briefly summarizes the research work of every chapter of this thesis. The following sections illustrate the thesis' novel works and solutions in terms of modeling, simulating, and optimizing data-driven applications in SDN-aware environments. The last section demonstrates a number of research directions that have been identified during the PhD studies. Every research direction encounters new challenges, which require new research works to be conducted.

7.1 Thesis Summary

The emerging of SDN paradigm has shitted the directions of using hard-to-program traditional networks to smart, automated networks. Such smart transition provides the ability of programming and controlling networks in real-time through a combination of SDN-aware applications and functions. As a result, SDN has been leveraged in several domains (e.g., edge datacenters, cloud datacenters) to obtain unprecedented network features along with improving cross-layer performance. Still, SDN-aware solutions considering different application and environment contexts are required due to the differences in dependencies across multiple levels of SDN-aware ecosystems along with different QoS requirements. As such, novel cross-layer optimization techniques, which is defined as NP-hard problems, are essential to obtain optimal solutions considering a set of variables and constraints.

To this end, this thesis explores several challenges in terms of developing novel modeling, simulation, and optimization techniques for several SDN-aware environments and applications, in addition to proposing novel solutions. The contributions of this is summarize as follows:

Chapter 2 first provides an overview of SDN and its architecture. It then demonstrates SDN deployment within diverse environments and applications. It next explains the different ways of testing and evaluating new SDN-aware designs, architectures, and optimization solutions. Based on illustrated research gaps and challenges, the chapter highlights the thesis' proposed solutions considering a set of models, algorithms,

and frameworks that facilitate the evaluation and testing of SDN-aware solutions in simulated environments.

Chapter 3 introduces a holistic modeling and integration of MapReduce BDMS-based applications in SDN-aware cloud infrastructures. As a result of such modeling, the chapter presents a novel simulation toolkit named "BigDataSDNSim," which provides a simulated infrastructure for researchers to quantify the performance impacts of MapReduce applications in terms of cross-layer optimization. A variety of application-network policies for diverse purposes (e.g., scheduling and routing) are illustrated, which helps implementing new cross-layer optimization solutions without a deep understanding of the complex interactions among BigDataSDNSim's components. Further, the validation and accuracy of the simulator are illustrated by comparing its output results with a real MapReduce SDN-aware environment. The practicality and advantages of using BigDataSDNSim are demonstrated by presenting two use cases, which show the impact of MapReduce-HDFS replication mechanisms and the advantages of using SDN.

Chapter 4 proposes a novel framework that facilitates the modeling, simulating, and evaluating of new algorithms, policies, and designs in the context of SD-WAN ecosystems and SDN-aware multiple cloud datacenters. The framework is implemented into a new simulator named "IoTSim-SDWAN," which is capable of providing a variety of modeling approaches and functionalities to evaluate and test SD-WAN cloud-based solutions. The chapter then demonstrates several proposed models, including SD-WAN ecosystems and TCP and UDP protocols. Further, the chapter presents network routing models of SD-WAN and classical WAN using graph theory, in addition to demonstrating new proposed coordination scheme for SD-WAN and SDN controllers residing in different datacenters. The chapter also illustrates the validation and correctness of the simulator in terms of measuring the level of similarities with a real-world network environment. Finally, a number of evaluation experiments with a goal to illustrate the practicality and features of IoTSim-SDWAN are presented.

Chapter 5 presents a new SDN-aware workflow broker that deploys the workflows of data-driven applications across multiple SD-WAN-enabled cloud datacenters. The chapter proposes an adaptive genetic algorithm to find solutions that maximizes the proportion of renewable energy usage and minimizes the real electricity cost along with

meeting several QoS constraints (e.g., user's given budget and deadline). Further, the chapter demonstrates several models used in the proposed approach, in addition to describing the proposed algorithm in detail. The chapters illustrates the performance of the algorithm using real data-driven workflows with different sizes under various configurations of VMs. Finally extensive experimental evaluation to study the feasibility of the proposed scheduling algorithm and architecture is presented.

Chapter 6 chapter presents a new framework that models the osmotic computing paradigm, which consists of heterogeneous, integrated SDN-aware edge-cloud environments along with the complexity of IoT applications interconnecting via SD-WAN. The framework is implemented into a new toolkit named IoTSim-Osmosis– a simulator that that enables the testing and evaluation of osmotic computing cross-layer design and optimization solutions in a unified modeling manner. The chapter demonstrates the IoTSim-Osmosis's proposed models in terms of osmotic application and network topologies and heterogeneous components dependencies. Finally, the chapters demonstrates the validation and efficacy of IoTSim-Osmosis by presenting a case study for an electricity management and billing osmotic application within a smart city.

7.2 Future Research Directions

During the phase of conducting this thesis, a number of research directions have been identified. The directions can be categorized into three areas: modeling and simulation, cross-layer optimization, and machine learning. Each area encounters new challenges and yet requires new research works to be conducted.

7.2.1 Modeling and simulation

In the current work, a number of simulation frameworks are proposed which model and simulate several SDN-aware ecosystems. The frameworks can be enhanced in a number of ways. As our future work, we will enhance BigDataSDNSim by adding big data stream models in the context SDN-aware multi cloud environments. We will also enhance IoTSim-SDWAN by modeling and implementing Network Function Virtualization (NFV) and a distributed application layer over multiple cloud datacenters. For

IoTSim-Osmosis, we would enhance it by modeling and implementing complex IoT protocols (e.g., XMPP). We will also enhance the current, basic wireless communications (e.g., WiFi) models by adding advanced models that simulates the mechanisms of different signal factors, such as distance and IoT device mobility.

7.2.2 Cross-layer optimization

In the current work, a number of algorithms are proposed to enhance the overall performance applications running SDN-aware ecosystems. Every proposed simulation framework is capable of testing and evaluating different context of problems and environments. Therefore, We will work on proposing a number of cross-layer optimization algorithms. In particular, we would focus on optimizing the deployment of MapReduce applications in SDN-aware cloud datacenters. We will also work on optimizing the deployment of osmotic applications considering coordination and data knowledge from several layers, including IoT, SDN-edge, SDN-cloud, and SD-WAN.

7.2.3 Machine learning

Simulation framework are capable of producing a massive amount of synthetic data according to given system models and hypotheses. In this thesis, every proposed simulation framework is also capable of producing synthetic data (e.g., traffic and processing matrices), which can be used in tackling different research problems other than in optimization contexts. For example, machine learning approaches and algorithms can use simulators' synthetic data to diagnose, predict, and detect the performance and behavior of different SDN ecosystems. As such, we would work on proposing machine learning solutions to predict and detect the performance degradation in MapReduce and osmotic applications.

References

[1] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[2] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya, "Cloudsimsdn: modeling and simulation of software-defined cloud data centers," in *Cluster, Cloud and Grid Computing (CCGrid), 15th IEEE/ACM International Symposium on*, Shenzhen, China, 2015.

[3] P. Qin, B. Dai, B. Huang, and G. Xu, "Bandwidth-aware scheduling with sdn in hadoop: A new trend for big data," *IEEE Systems Journal*, vol. 11, no. 4, pp. 2337–2344, 2015.

[4] S. Zhao, A. Sydney, and D. Medhi, "Building application-aware network environments using sdn for optimizing hadoop applications," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 583–584.

[5] M. Tüxen, I. Rüngeler, and E. P. Rathgeb, "Interface connecting the inet simulation framework with the real world," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008, pp. 1–6.

[6] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of sdn/openflow controllers," in *Proceedings of the 9th central & eastern european software engineering conference in russia*, 2013, pp. 1–6.

[7] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[8] J. Son and R. Buyya, "A taxonomy of software-defined networking (sdn)-enabled cloud computing," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–36, 2018.

[9] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[10] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic sdn controller assignment in data center networks: Stable matching with transfers," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[11] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2359–2391, 2017.

[12] L. Cui, F. R. Yu, and Q. Yan, "When big data meets software-defined networking: Sdn for big data and big data for sdn," *IEEE network*, vol. 30, no. 1, pp. 58–65, 2016.

[13] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Ranjan, "Optimal decision making for big data processing at edge-cloud environment: An sdn perspective," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 2, pp. 778–789, 2017.

[14] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang, "In-memory big data management and processing: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1920–1948, 2015.

[15] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, and A. Bouras, "A survey of clustering algorithms for big data: Taxonomy and empirical analysis," *IEEE transactions on emerging topics in computing*, vol. 2, no. 3, pp. 267–279, 2014.

[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," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 3–14.

[17] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 15–26.

[18] R. Cahn, *Wide area network design: concepts and tools for optimization*. Morgan Kaufmann, 1998.

[19] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (sd-wan): Architecture, advances and opportunities," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019, pp. 1–9.

[20] X. Li, D. Li, J. Wan, C. Liu, and M. Imran, "Adaptive transmission optimization in sdn-based industrial internet of things with edge computing," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1351–1360, 2018.

[21] M. Uddin, S. Mukherjee, H. Chang, and T. Lakshman, "Sdn-based multi-protocol edge switching for iot service automation," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2775–2786, 2018.

[22] C. Li, Y. Xue, J. Wang, W. Zhang, and T. Li, "Edge-oriented computing paradigms: A survey on architecture design and system management," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–34, 2018.

[23] R. Muñoz, L. Nadal, R. Casellas, M. S. Moreolo, R. Vilalta, J. M. Fàbrega, R. Martínez, A. Mayoral, and F. J. Vílchez, "The adrenaline testbed: An sdn/nfv packet/optical transport network and edge/core cloud platform for end-to-end 5g and iot services," in *2017 European Conference on Networks and Communications (EuCNC)*. IEEE, 2017, pp. 1–5.

[24] Z. Lv and W. Xiu, "Interaction of edge-cloud computing based on sdn and nfv for next generation iot," *IEEE Internet of Things Journal*, 2019.

[25] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, 2016.

[26] J. Son, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Sla-aware and energy-efficient dynamic overbooking in sdn-based cloud data centers," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 76–89, 2017.

[27] J. Chase, R. Kaewpuang, W. Yonggang, and D. Niyato, "Joint virtual machine and bandwidth allocation in software defined network (sdn) and cloud computing environments," in *2014 IEEE international conference on communications (ICC)*. IEEE, 2014, pp. 2969–2974.

[28] D. A. Chekired, M. A. Togou, L. Khoukhi, and A. Ksentini, "5g-slicing-enabled scalable sdn core network: Toward an ultra-low latency of autonomous driving service," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1769–1782, 2019.

[29] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE communications surveys & tutorials*, vol. 18, no. 1, pp. 602–622, 2015.

[30] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," 2004.

[31] M. R. Garey and D. S. Johnson, "A guide to the theory of np-completeness," *Computers and Intractability*, pp. 37–79, 1990.

[32] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 473–478, 2012.

[33] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu, "The sdn controller placement problem for wan," in *2014 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, 2014, pp. 220–224.

[34] M. Amiri, H. Al Osman, and S. Shirmohammadi, "Game-aware and sdn-assisted bandwidth allocation for data center networks," in *2018 IEEE conference on multimedia information processing and retrieval (MIPR)*. IEEE, 2018, pp. 86–91.

[35] A. Beznosyk, P. Quax, K. Coninx, and W. Lamotte, "Influence of network delay and jitter on cooperation in multiplayer games," in *Proceedings of the 10th international conference on virtual reality continuum and its applications in industry*, 2011, pp. 351–354.

[36] M. Amiri, A. Sobhani, H. Al Osman, and S. Shirmohammadi, "Sdn-enabled game-aware routing for cloud gaming datacenter network," *IEEE Access*, vol. 5, pp. 18 633–18 645, 2017.

[37] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud task scheduling based on ant colony optimization," in *2013 8th international conference on computer engineering & systems (ICCES)*. IEEE, 2013, pp. 64–69.

[38] A. K. Singh and S. Srivastava, "A survey and classification of controller placement problem in sdn," *International Journal of Network Management*, vol. 28, no. 3, p. e2018, 2018.

[39] S. Zhao and D. Medhi, "Application-aware network design for hadoop mapreduce optimization using software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 804–816, 2017.

[40] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[41] M. Casado and N. McKeown, "The virtual network system," in *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, 2005, pp. 76–80.

[42] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[43] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *2009 Fifth International Joint Conference on INC, IMS and IDC*. Ieee, 2009, pp. 44–51.

[44] M. Zhou, R. Zhang, D. Zeng, and W. Qian, "Services in the cloud computing era: A survey," in *2010 4th International Universal Communication Symposium*. IEEE, 2010, pp. 40–46.

[45] M. Chen, S. Mao, and Y. Liu, "Big data: a survey," *Mobile networks and applications*, vol. 19, no. 2, pp. 171–209, 2014.

[46] K. Kambatla, G. Kollias, V. Kumar, and A. Grama, "Trends in big data analytics," *Journal of Parallel and Distributed Computing*, vol. 74, no. 7, pp. 2561–2573, 2014.

[47] M. H. Iqbal and T. R. Soomro, "Big data analysis: Apache storm perspective," *International journal of computer trends and technology*, vol. 19, no. 1, pp. 9–14, 2015.

[48] R. Ranjan, "Streaming big data processing in datacenter clouds," *IEEE Cloud Computing*, vol. 1, no. 1, pp. 78–83, 2014.

[49] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, Santa Clara, California, USA, 2013.

[50] M. Wyle, "A wide area network information filter," in *Proceedings First International Conference on Artificial Intelligence Applications on Wall Street.* IEEE, 1991, pp. 10–15.

[51] K. Alwasel, Y. Li, P. P. Jayaraman, S. Garg, R. N. Calheiros, and R. Ranjan, "Programming sdn-native big data applications: research gap analysis," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 62–71, 2017.

[52] C. Yu, J. Lan, Z. Guo, Y. Hu, and T. Baker, "An adaptive and lightweight update mechanism for sdn," *IEEE Access*, vol. 7, pp. 12 914–12 927, 2019.

[53] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE access*, vol. 6, pp. 6900–6919, 2017.

[54] S. H. Shah and I. Yaqoob, "A survey: Internet of things (iot) technologies, applications and challenges," in *2016 IEEE Smart Energy Grid Engineering (SEGE).* IEEE, 2016, pp. 381–385.

[55] J. Lloret, J. Tomas, A. Canovas, and L. Parra, "An integrated iot architecture for smart metering," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 50–57, 2016.

[56] D. N. Jha, P. Michalak, Z. Wen, P. Watson, and R. Ranjan, "Multi-objective deployment of data analysis operations in heterogeneous iot infrastructure," *IEEE Transactions on Industrial Informatics*, 2019.

[57] A. Ahmed and A. S. Sabyasachi, "Cloud computing simulators: A detailed survey and future direction," in *2014 IEEE international advance computing conference (IACC).* IEEE, 2014, pp. 866–872.

[58] K. Fall, "Network emulation in the vint/ns simulator," in *Proceedings IEEE International Symposium on Computers and Communications (Cat. No. PR00250).* IEEE, 1999, pp. 244–250.

[59] S.-Y. Wang, "Comparison of sdn openflow network simulator and emulators: Estinet vs. mininet," in *2014 IEEE Symposium on Computers and Communications (ISCC).* IEEE, 2014, pp. 1–6.

[60] K. Wehrle, M. Günes, and J. Gross, *Modeling and tools for network simulation.* Springer Science & Business Media, 2010.

[61] G. I. Hawe, G. Coates, D. T. Wilson, and R. S. Crouch, "Agent-based simulation for large-scale emergency response: A survey of usage and implementation," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–51, 2012.

[62] F. Persson and J. Olhager, "Performance simulation of supply chain designs," *International journal of production economics*, vol. 77, no. 3, pp. 231–245, 2002.

[63] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar, "Tarazu: optimizing mapreduce on heterogeneous clusters," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 61–74, 2012.

[64] M. J. Fischer, X. Su, and Y. Yin, "Assigning tasks for efficiency in hadoop," in *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*, 2010, pp. 30–39.

[65] D. Cheng, J. Rao, C. Jiang, and X. Zhou, "Resource and deadline-aware job scheduling in dynamic hadoop clusters," in *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2015, pp. 956–965.

[66] I. A. T. Hashem, N. B. Anuar, M. Marjani, E. Ahmed, H. Chiroma, A. Firdaus, M. T. Abdullah, F. Alotaibi, W. K. M. Ali, I. Yaqoob *et al.*, "Mapreduce scheduling algorithms: a review," *The Journal of Supercomputing*, pp. 1–31, 2018.

[67] A.-C. G. Anadiotis, G. Morabito, and S. Palazzo, "An sdn-assisted framework for optimal deployment of mapreduce functions in wsns," *IEEE Transactions on Mobile Computing*, vol. 15, no. 9, pp. 2165–2178, 2015.

[68] S. K. Garg and R. Buyya, "Networkcloudsim: modelling parallel applications in cloud simulations," in *Utility and Cloud Computing (UCC), Fourth IEEE International Conference on*, Victoria, NSW, Australia, 2011.

[69] W. Chen and E. Deelman, "Workflowsim: a toolkit for simulating scientific workflows in distributed environments," in *E-science (e-science), IEEE 8th International Conference on*, Chicago, IL, USA, 2012.

[70] D. Kliazovich, P. Bouvry, and S. U. Khan, "Greencloud: a packet-level simulator of energy-aware cloud computing data centers," *The Journal of Supercomputing*, vol. 62, no. 3, pp. 1263–1283, 2012.

[71] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "icancloud: a flexible and scalable cloud infrastructure simulator," *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, 2012.

[72] B. Lantz, B. Heller, and 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*, Monterey, California, USA, 2010.

[73] P. Wette, M. Dräxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "Maxinet: distributed emulation of software-defined networks," in *2014 IFIP Networking Conference*. IEEE, 2014, pp. 1–9.

[74] M. V. Neves, C. A. De Rose, and K. Katrinis, "Mremu: an emulation-based framework for datacenter network experimentation using realistic mapreduce traffic," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE 23rd International Symposium on*, Atlanta, GA, USA, 2015.

[75] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, "Estinet openflow network simulator and emulator," *IEEE Communications Magazine*, vol. 51, no. 9, pp. 110–117, 2013.

[76] P. Kathiravelu and L. Veiga, "Software-defined simulations for continuous development of cloud and data center networks," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2016, pp. 3–23.

[77] S. Ghosh, S. Busari, T. Dagiuklas, M. Iqbal, R. Mumtaz, J. Gonzalez, S. Stavrou, and L. Kanaris, "Sdn-sim: integrating a system-level simulator with a software defined network," *IEEE Communications Standards Magazine*, vol. 4, no. 1, pp. 18–25, 2020.

[78] X. Zeng, S. K. Garg, P. Strazdins, P. P. Jayaraman, D. Georgakopoulos, and R. Ranjan, "Iotsim: a simulator for analysing iot applications," *Journal of Systems Architecture*, vol. 72, pp. 93–107, 2017.

[79] J. Jung and H. Kim, "Mr-cloudsim: designing and implementing mapreduce computing model on cloudsim," in *ICT Convergence (ICTC), International Conference on*, Jeju Island, South Korea, 2012.

[80] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu, "Mrsim: a discrete event based mapreduce simulator," in *Fuzzy Systems and Knowledge Discovery (FSKD), Seventh International Conference on*, Yantai, China, 2010.

[81] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "Using realistic simulation for performance analysis of mapreduce setups," in *Proceedings of the 1st ACM workshop on Large-Scale system and application performance*, Garching, Germany, 2009.

[82] L. B. de Almeida, E. C. de Almeida, J. Murphy, E. Robson, and A. Ventresque, "Bigdatanetsim: a simulator for data and process placement in large big data platforms," in *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 2018, pp. 1–10.

[83] C. Calcaterra, A. Carmenini, A. Marotta, U. Bucci, and D. Cassioli, "Maxhadoop: an efficient scalable emulation tool to test sdn protocols in emulated hadoop environments," *Journal of Network and Systems Management*, pp. 1–29, 2020.

[84] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: characterization of the mapreduce-based data analysis," in *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*. IEEE, 2010, pp. 41–51.

[85] B. Pfaff and B. Davie, "The open vswitch database management protocol," *Internet Requests for Comments, RFC Editor, RFC*, vol. 7047, 2013.

[86] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[87] F. Tomonori, "Introduction to ryu sdn framework," *Open Networking Summit*, pp. 1–14, 2013.

[88] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.

[89] N. Abbas, M. Asim, N. Tariq, T. Baker, and S. Abbas, "A mechanism for securing iot-enabled applications at the fog layer," *Journal of Sensor and Actuator Networks*, vol. 8, no. 1, p. 16, 2019.

[90] M. Al-Khafajiy, T. Baker, A. Waraich, D. Al-Jumeily, and A. Hussain, "Iot-fog optimal workload via fog offloading," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018, pp. 359–364.

[91] M. Al-khafajiy, T. Baker, M. Asim, Z. Guo, R. Ranjan, A. Longo, D. Puthal, and M. Taylor, "Comitment: A fog computing trust management approach," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 1–16, 2020.

[92] "aryaka," μhttps://www.aryaka.com/sd-wan-as-a-service/, accessed October 30, 2019.

[93] "silver-peak," μhttps://www.silver-peak.com/, accessed October 30, 2019.

[94] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," Tech. Rep., 2000.

[95] K. Alwasel, R. N. Calheiros, S. Garg, R. Buyya, M. Pathan, D. Georgakopoulos, and R. Ranjan, "Bigdatasdnsim: A simulator for analyzing big data applications in software-defined cloud data centers," *Software: Practice and Experience*, 2020.

[96] B. Lantz, B. Heller, and 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*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: μhttp://doi.acm.org/10.1145/1868447.1868466

[97] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34. [Online]. Available: μhttps://doi.org/10.1007/978-3-642-12331-3_2

[98] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, p. e3493, 2018.

[99] 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," *Software: Practice and Experience*.

[100] C. M. Kozierok, *The TCP/IP guide: a comprehensive, illustrated Internet protocols reference*. No Starch Press, 2005.

[101] T. Kelly, "Scalable tcp: Improving performance in highspeed wide area networks," *ACM SIGCOMM computer communication Review*, vol. 33, no. 2, pp. 83–91, 2003.

[102] A. B. Forouzan, *Data communications & networking*. Tata McGraw-Hill Education, 2007.

[103] "Wireshark," μhttps://www.wireshark.org/, accessed July 22, 2019.

[104] V. Jacobson, "Compressing tcp/ip headers for low-speed serial links," 1990.

[105] "The internet topology zoo," μhttp://www.topology-zoo.org, accessed July 24, 2019.

[106] "Shenzhen helor cloud computer," μhttp://www.hlypc.com/, accessed July 24, 2019.

[107] "Open vswitch," μ https://www.openvswitch.org/, accessed July 24, 2019.

[108] "Ryu sdn framework," μ https://osrg.github.io/ryu/, accessed July 24, 2019.

[109] "iperf - the ultimate speed test tool for tcp, udp and sctp," μhttps://iperf.fr, accessed July 24, 2019.

[110] R. Ramaswamy, N. Weng, and T. Wolf, "Characterizing network processing delay," in *IEEE Global Telecommunications Conference, 2004. GLOBECOM'04.*, vol. 3. IEEE, 2004, pp. 1629–1634.

[111] F. Tao, Q. Qi, A. Liu, and A. Kusiak, "Data-driven smart manufacturing," *Journal of Manufacturing Systems*, vol. 48, pp. 157–169, 2018.

[112] C. Jiang, Y. Chen, Q. Wang, and K. R. Liu, "Data-driven auction mechanism design in iaas cloud computing," *IEEE Transactions on Services Computing*, vol. 11, no. 5, pp. 743–756, 2015.

[113] J. Koomey *et al.*, "Growth in data center electricity use 2005 to 2010," *A report by Analytical Press, completed at the request of The New York Times*, vol. 9, p. 161, 2011.

[114] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, 2020.

[115] M. R. Bonyadi, Z. Michalewicz, and L. Barone, "The travelling thief problem: The first step in the transition from theoretical problems to realistic problems," in *2013 IEEE Congress on Evolutionary Computation*. IEEE, 2013, pp. 1037–1044.

[116] Z. Wen, J. Cała, P. Watson, and A. Romanovsky, "Cost effective, reliable and secure workflow deployment over federated clouds," *IEEE Transactions on Services Computing*, vol. 10, no. 6, pp. 929–941, 2016.

[117] Z. Wen, R. Qasha, Z. Li, R. Ranjan, P. Watson, and A. Romanovsky, "Dynamically partitioning workflow over federated clouds for optimising the monetary cost and handling run-time failures," *IEEE Transactions on Cloud Computing*, 2016.

[118] H. Yuan, J. Bi, M. Zhou, and K. Sedraoui, "Warm: Workload-aware multi-application task scheduling for revenue maximization in sdn-based cloud data center," *IEEE Access*, vol. 6, pp. 645–657, 2017.

[119] J. Wu, S. Guo, J. Li, and D. Zeng, "Big data meet green challenges: Greening big data," *IEEE Systems Journal*, vol. 10, no. 3, pp. 873–887, 2016.

[120] H. Yuan, J. Bi, M. Zhou, and A. C. Ammari, "Time-aware multi-application task scheduling with guaranteed delay constraints in green data center," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 3, pp. 1138–1151, 2017.

[121] J. Bi, H. Yuan, W. Tan, M. Zhou, Y. Fan, J. Zhang, and J. Li, "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 1172–1184, 2015.

[122] K. Vahi, I. Harvey, T. Samak, D. Gunter, K. Evans, D. Rogers, I. Taylor, M. Goode, F. Silva, E. Al-Shakarchi *et al.*, "A case study into using common real-time workflow monitoring infrastructure for scientific workflows," *Journal of grid computing*, vol. 11, no. 3, pp. 381–406, 2013.

[123] M. Giacobbe, A. Celesti, M. Fazio, M. Villari, and A. Puliafito, "Evaluating a cloud federation ecosystem to reduce carbon footprint by moving computational resources," in *2015 IEEE Symposium on Computers and Communication (ISCC)*, July 2015, pp. 99–104.

[124] D. Bhandari, C. Murthy, and S. K. Pal, "Genetic algorithm with elitist model and its convergence," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 10, no. 06, pp. 731–747, 1996.

[125] R. Poli and W. B. Langdon, "Genetic programming with one-point crossover and point mutation," in *Soft Computing in Engineering Design and Manufacturing*. Springer-Verlag, 1997, pp. 180–189.

[126] D. E. Golberg, "Genetic algorithms in search, optimization, and machine learning," *Addion wesley*, vol. 1989, p. 102, 1989.

[127] J. Smith and T. C. Fogarty, "Self adaptation of mutation rates in a steady state genetic algorithm," in *Proceedings of IEEE international conference on evolutionary computation.* IEEE, 1996, pp. 318–323.

[128] 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)," *Journal of Parallel and Distributed Computing*, 2020.

[129] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[130] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," *Future Generation Computer Systems*, vol. 48, pp. 1–18, 2015.

[131] H. Yuan, J. Bi, W. Tan, M. Zhou, B. H. Li, and J. Li, "Ttsa: An effective scheduling approach for delay bounded tasks in hybrid clouds," *IEEE transactions on cybernetics*, vol. 47, no. 11, pp. 3658–3668, 2016.

[132] G. S. Aujla and N. Kumar, "Sdn-based energy management scheme for sustainability of data centers: An analysis on renewable energy sources and electric vehicles participation," *Journal of Parallel and Distributed Computing*, vol. 117, pp. 228–245, 2018.

[133] ——, "Mensus: An efficient scheme for energy management with sustainability of cloud data centers in edge–cloud environment," *Future Generation Computer Systems*, vol. 86, pp. 1279–1300, 2018.

[134] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, Nov 2011, pp. 1–12.

[135] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 22:1–22:11.

[136] R. N. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1787–1796, 2013.

[137] M. R. H. Farahabady, Y. C. Lee, and A. Y. Zomaya, "Pareto-optimal cloud bursting," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2670–2682, 2013.

[138] H. M. Fard, R. Prodan, and T. Fahringer, "A truthful dynamic workflow scheduling mechanism for commercial multicloud environments," *IEEE Transactions on Parallel and Distributed systems*, vol. 24, no. 6, pp. 1203–1212, 2012.

[139] F. Jrad, J. Tao, I. Brandic, and A. Streit, "{SLA} enactment for large-scale healthcare workflows on multi-cloud," *Future Generation Computer Systems*, vol. 43–44, no. 0, pp. 135 – 148, 2015.

[140] M. Catena and N. Tonellotto, "Energy-efficient query processing in web search engines," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 7, pp. 1412–1425, 2017.

[141] Y. Li, A.-C. Orgerie, I. Rodero, B. L. Amersho, M. Parashar, and J.-M. Menaud, "End-to-end energy models for edge cloud-based iot platforms: Application to data stream analysis in iot," *Future Generation Computer Systems*, vol. 87, pp. 667–678, 2018.

[142] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing, "Utilizing green energy prediction to schedule mixed batch and service jobs in data centers," *ACM SIGOPS Operating Systems Review*, vol. 45, no. 3, pp. 53–57, 2012.

[143] Í. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenhadoop: leveraging green energy in data-processing frameworks," in *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 2012, pp. 57–70.

[144] W. Deng, F. Liu, H. Jin, C. Wu, and X. Liu, "Multigreen: Cost-minimizing multi-source datacenter power supply with online control," in *Proceedings of the fourth international conference on Future energy systems*. ACM, 2013, pp. 149–160.

[145] S. K. Garg, C. S. Yeo, and R. Buyya, "Green cloud framework for improving carbon efficiency of clouds," in *European Conference on Parallel Processing*. Springer, 2011, pp. 491–502.

[146] R. T. Kaushik, L. Cherkasova, R. Campbell, and K. Nahrstedt, "Lightning: self-adaptive, energy-conserving, multi-zoned, commodity green cloud storage system," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 332–335.

[147] A. Kiani and N. Ansari, "Toward low-cost workload distribution for integrated green data centers," *IEEE Communications Letters*, vol. 19, no. 1, pp. 26–29, 2015.

[148] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): a vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[149] M. Tao, J. Zuo, Z. Liu, A. Castiglione, and F. Palmieri, "Multi-layer cloud architectural model and ontology-based security service framework for iot-based

smart homes," *Future Generation Computer Systems*, vol. 78, pp. 1040–1051, 2018.

[150] I. Lee and K. Lee, "The internet of things (iot): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[151] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and 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.

[152] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[153] M. Villari, M. Fazio, S. Dustdar, O. Rana, D. N. Jha, and R. Ranjan, "Osmosis: The osmotic computing platform for microelements in the cloud, edge, and internet of things," *Computer*, vol. 52, no. 8, pp. 14–26, 2019.

[154] D. Citron and A. Zlotnick, "Testing large-scale cloud management," *IBM Journal of Research and Development*, vol. 55, no. 6, pp. 6–1, 2011.

[155] M. Tighe, G. Keller, M. Bauer, and 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.

[156] W. Lin, S. Xu, L. He, and J. Li, "Multi-resource scheduling and power simulation for cloud computing," *Information Sciences*, vol. 397, pp. 168–186, 2017.

[157] G. Kecskemeti, "Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds," *Simulation Modelling Practice and Theory*, vol. 58, pp. 188–218, 2015.

[158] M. Afanasov, L. Mottola, and C. Ghezzi, "Software adaptation in wireless sensor networks," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 12, no. 4, pp. 1–29, 2018.

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

[160] P. Levis, N. Lee, M. Welsh, and 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.

[161] 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.

[162] A. Varga and 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.

[163] 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.

[164] D. Benedetti, C. Petrioli, and 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.

[165] S. Sotiriadis, N. Bessis, E. Asimakopoulou, and 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.

[166] N. Mohan and 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.

[167] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[168] M. M. Lopes, W. A. Higashino, M. A. Capretz, and 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.

[169] 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," *Software: Practice and Experience*, vol. 50, no. 6, pp. 844–867, 2020.

[170] B. Bertran, J. Bruneau, D. Cassou, N. Loriant, E. Balland, and C. Consel, "Diasuite: A tool suite to develop sense/compute/control applications," *Science of Computer Programming*, vol. 79, pp. 39–51, 2014.

[171] S. Chauhan, P. Patel, A. Sureka, F. C. Delicato, and 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.

[172] M. Martin and 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.