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

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

Khaled Alwasel[a,b], Devki Nandan Jha[a], Eduardo Hernandez[a], Deepak Puthal[a], Mutaz Barika[c], Blesson Varghese[d], Saurabh Kumar Garg[c], Philip James[a], Albert Zomaya[e], Graham Morgan[a], Rajiv Ranjan[a]

[a]*Newcastle University, Newcastle upon Tyne, UK*
[b]*Saudi Electronic University, Riyadh, Saudi Arabia*
[c]*University of Tasmania, Hobart, Australia*
[d]*Queen's University, Belfast, UK*
[e]*The University of Sydney, Sydney, Australia*

Abstract

Software-defined networking (SDN) has evolved as an approach that allows network administrators to program and initialise, control, change and manage networking components (mostly at L2-L3 layers) of the OSI model. SDN is designed to address the programmability shortcomings of traditional networking architectures commonly used in cloud datacenters (CDC). Deployment of SDN solutions have demonstrated significant improvements in areas such as flow optimisation and bandwidth allocation in a CDC. However, the benefits are significantly less explored when considering Software-Defined Wide Area Networks (SD-WAN) architectures in the context of delivering solutions by networking multiple CDCs. To support the testing and benchmarking of data-driven applications that rely on data ingestion and processing (e.g., Smart Energy Cloud, Content Delivery Networks) across multiple cloud datacenters, this paper presents the simulator, 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-enabled multiple cloud datacenters. Finally, IoTSim-SDWAN simulator 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.

Keywords: , Software-Defined Wide Area Network (SD-WAN), Software-Defined Network (SDN), Classical WAN, Internet of Things (IoT)

1. Introduction

Proliferation of cloud computing has revolutionized hosting and delivery of Internet-based application services that rely on managing real-time streaming data (e.g., smart energy cloud solutions based on smart meters connected to millions of households) [1, 2]. As streaming data sources (e.g., smart meters, smart thermostats, sensors) are geographically distributed, the network quality of service (QoS) (data transfer latency) for data ingestion and processing varies [3]. This variation is dependent upon the location of cloud datacenters CDCs in relation to the varied locations of input data streams. The main reason for variable network QoS across CDCs is the underlying TCP/IP based wide area networking (WAN) architectures that satisfy networking requirements across the CDC/sensor infrastructure.

WANs are the core communication infrastructures that interconnect geographically distributed systems and devices into a single network [4]. 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 CDC contected solution affording resource management and efficiency within modern applications (e.g., smart energy clouds, content delivery network, distributed gaming), the WAN needs to be part of the whole adaptable SDN solution.

Recently, Software-Defined Wide Area Network (SD-WAN) has emerged as a promising solution to alleviate issues that inhibit the use of static WAN deployment for smart applications deployment [5, 6, 7]. SD-WAN originates from the Software-Defined Network (SDN) paradigm, proposing SDN's mechanisms of managing, operating, automating, and simplifying networks within a WAN context [8]. The concept of SDN, in terms of decoupling the control and data planes, is applied to SD-WAN ecosystem in addition to leveraging software-based centralized controllers. SDN is primarily responsible for

controlling and managing the internal network operations of CDCs and local area networks (LANs) whereas SD-WAN is moves the focus to also include managing the interconnecting applications spanning different CDCs.

While significant research has been achieved in proposing and evaluating solutions for a SDN located in a CDC, there is a need to address the shortfall in proposing and evaluating new SD-WAN solutions. A way of determining the performance of SD-WAN solutions can be to monitor and evaluate the use of real-world SD-WAN infrastructures. However, this method of studying and analyzing any proposed SD-WAN solutions has a number of limitations: (1) requires access to real-world SD-WAN infrastructures that may be propriety and out of the reach of academic researchers, (2) observed results are error prone due to the dynamic and changing nature of SD-WAN, (3) implementing systems simply determine suitability is time-consuming and requires significant human resources, (4) real-world SD-WAN infrastructures suffer from scalability and flexibility issues in that they are application domain specific.

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 centre 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 modelling 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. Utilising simulations and associated models allows researchers and engineers to rapidly investigate, evaluate, and optimize their proposed solutions in a cost-effective and time-saving manner. This approach has been demonstrated successfully in many application domains set within a context of a distributed system. In the last few years, there are many simulation and emulation tools that have been developed to aid researchers and developers to evaluate new algo-

rithms for the management of different computing resources and systems in a controllable and repeatable manner, such as CloudSimSDN[9], Mininet[10], GreenCloud[11], and NetworkCloudSim[12]. However, these tools lack the modeling and simulation capability of modelling mutiple SDN-enabled datacenters running within SD-WAN environments. Contributing towards the process of SD-WAN investigation and development, we present a new simulation tool: IoTSim-SDWAN (Software-Defined Wide Area Network Simulator). This is a stand-alone Java-based tool that simulates SD-WAN ecosystems and SDN-enabled 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-enabled multi-cloud datacenters.

We empirically validate the models, formulas, and framework of IoTSim-SDWAN using real world network data. Our 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, we design two experiments to demonstrate the practicality and capability of IoTSim-SDWAN. The assumption is that classical WAN and cloud datacenters enable gateways and switches to have full control of their network decisions while SD-WAN and SDN-enabled environments enable SD-WAN and SDN controllers to have full network control to dynamically instruct gateways and switches in real-time. We therefore seek an evaluation objective 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.

In summary, the main contributions of this paper are as follows:

- Proposing a novel framework that simulates and models the SD-WAN and SDN-enabled 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 paper is divided into several sections as follows. Section 2 discusses design criteria and motivation. The design and modeling capabilities of IoTSim-SDWAN is presented in Section 3. Section 4 describes the empirical validation and accuracy of IoTSim-SDWAN. Experiments are presented in Section 5 to show how IoTSim-SDWAN can contribute to multi-CDC design issues for smart applications. Section 6 illustrates the most rel event related work. Section 7 concludes the paper and highlights our future plans.

2. Background

The (classical) WAN is a core communications layer and is 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% [6]). 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 [5] in 2013 followed by Google in 2014 [6]. 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 optimisations.

Figure 1 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 plane maintains routing information while the control plane is responsible for making all network decisions and providing the data plane with routing information. From Figure 1, we can clearly identify that the SD-WAN controller oversees and manages the whole

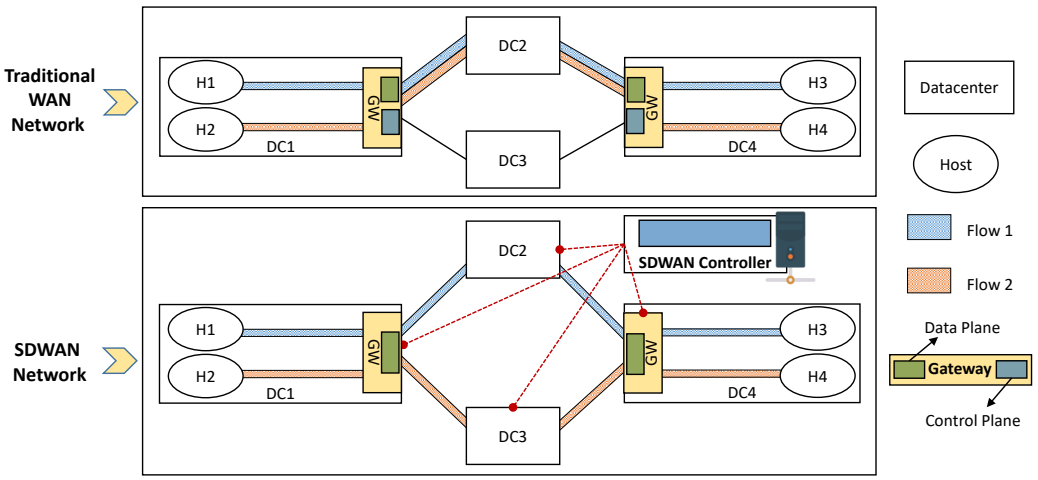


Figure 1: Key difference between classical WAN and SD-WAN networks

network. this results in cohesive global network decisions made with awareness of current (and legacy) traffic issues. The SD-WAN is also capable of enforcing a new network policy/QoS on the fly, something that a classical WAN may never achieve due to its static nature. An SD-WAN is also 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 utilization of resources).

A simplified view of the layered architecture of distributed enterprise ecosystems that utilise an SD-WAN is highlighted in Figure 2. 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 optimised 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).

A 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 [13], Silver-Peak [14]). 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 [6]) to manage

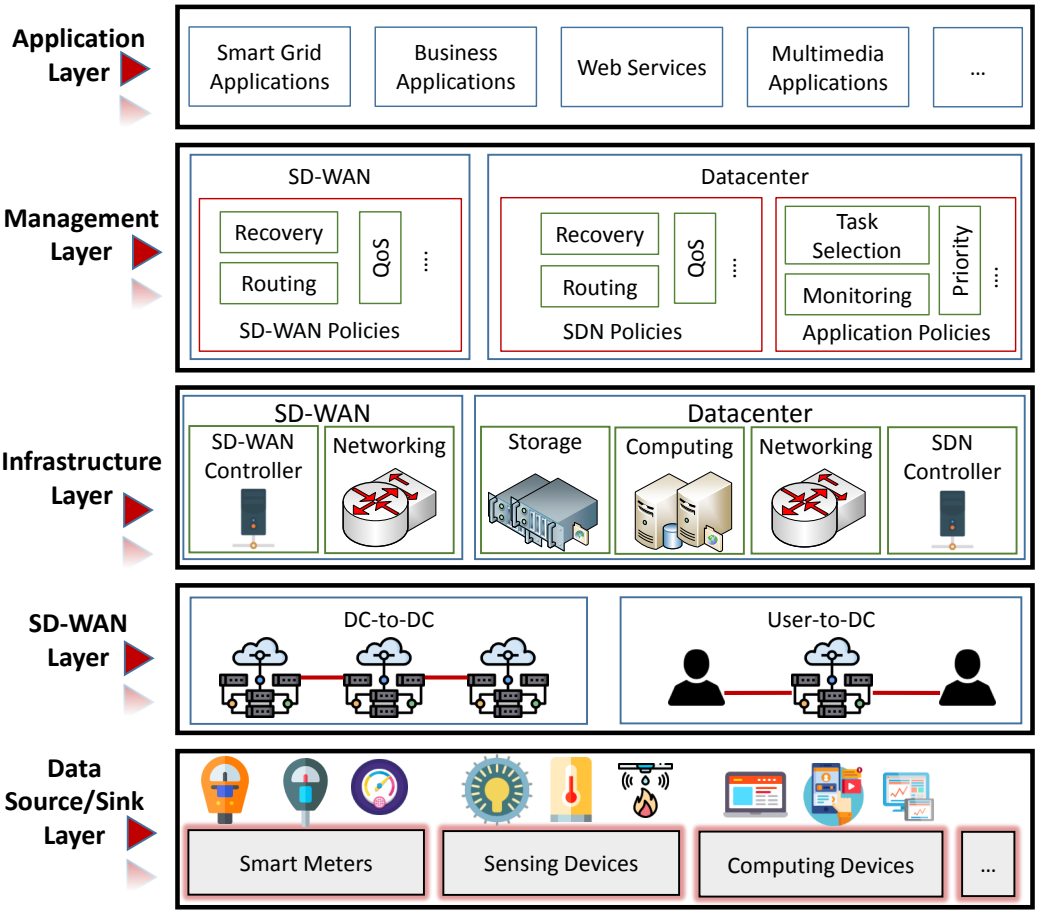


Figure 2: Architecture of distributed enterprise ecosystems and SD-WAN

their own resources in the context of mass date migration and storage requirements. The relation between enterprise ecosystems and SD-WAN is demonstrated in 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 a 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 datacenter-to-datacenter (DC-to-DC) and assumes responsibility for exchanging large amounts of data across geographically dispersed datacenters. Secondly, user-to-datacenter (user-to-DC) connects end users and datacenters. Both types require different management and policies for defining and modifying an SD-WAN in real-time based on changing traffic flow requirements. The current stage of our work presented in this paper (IoTSim-SDWAN) focuses on the modeling of DC-to-DC.

- *Infrastructure* consists of datacenter and SD-WAN hardware supporting requirements. 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 an SDN. The management layer also offers features to an 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. We support the modeling and evaluation of network performance and energy consumption.

3. Design of IoTSim-SDWAN

This section describes the model and framework designs of IoTSim-SDWAN. Sub-section 3.1 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 3.2. Sub-section 3.3 demonstrates the network modeling of SD-WAN and classical WAN based on graph theory along with presenting our proposed Shortest Path Map Based (SPMB) routing algorithm and also presents our proposed coordination scheme for SD-WAN and SDN controllers. Sub-section 3.4 illustrates the system structure overview and physical properties of IoTSim-SDWAN and also illustrates the interactions among the components of IoTSim-SDWAN in a simplified form.

3.1. 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

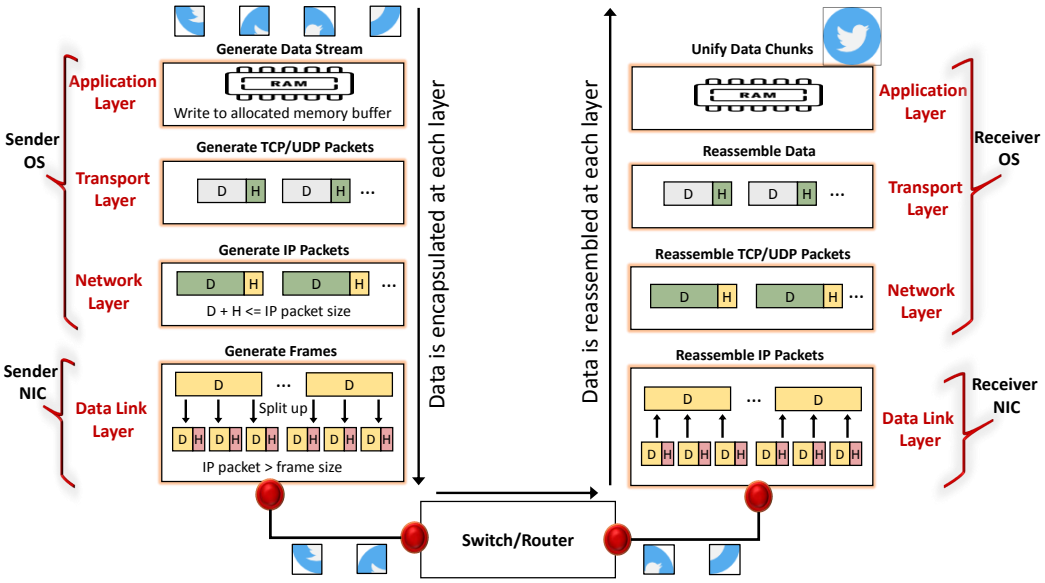


Figure 3: Procedure of data transmission based on the TCP/IP model

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 3, a conceptual network model (known as a TCP/IP model [15]) 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 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 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 head-

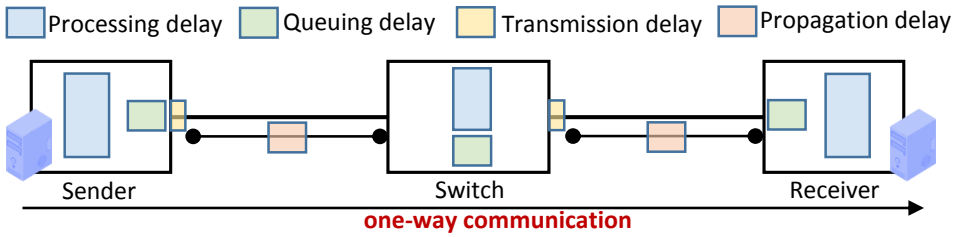


Figure 4: Overview of network delays

ers 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-enabled 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 3.2). 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). 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.

3.2. Theoretical model

The transport layer of TCP/IP model (as shown in figure 3) 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

Table 1: 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

size (a.k.a TCP window scaling [16]) 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 1 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 as in equation 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.

$$pn = \frac{ds(i)}{pl_s} \tag{1}$$

Each packet at both the transport and network layers must be tagged with a unique identification header. Equation 2 is used to obtain the header size $hs(p)$ of every packet p where tp_h is the transport header size and ip_h is the network (IP) header size. Equation 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 $hs(p)$

$$hs(p) = tp_h + ip_h \tag{2}$$

$$ts(p) = pl_s + hs(p), \forall p \in \{1, 2, ..., pn\} \tag{3}$$

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 [17]. 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 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 5 where $dl_{ht}(p)$ is header and footer sizes appended by the data link layer.

$$nf = \frac{\sum_{n=1}^{pn} ts(p_n)}{mtu_s} \tag{4}$$

$$tf = mtu_s + dl_{ht}(p) \tag{5}$$

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 6.

$$nc = \sum_{i,j \in H} c(i,j) \tag{6}$$

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 9). 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 7 is used to compute the total number of channels *tc(n)* carried out by a given nth link.

$$tc(n) = \sum_{c \in C} l(c) \tag{7}$$

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 8.

$$c_{bw}(i,j) = min\left(\frac{l_{bw}(n)}{tc(n)}\right) \tag{8}$$

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 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 10 is 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 11.

$$f_n = \sum_{f \in F} f(si, dj) \tag{9}$$

$$f_{bw} = \frac{c_{bw}}{f_n} \tag{10}$$

$$f_s = \sum_{j=1}^{nf} tf(j) \tag{11}$$

For every frame, there are transit delays encountered, as shown in figure 4. As described earlier in 3.1, 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 12 is used to compute all delays d. Equation 13 is used to obtain total delays t_d for all frames belonging to a given flow f.

$$d = d_p + d_q + d_t + d_\rho \tag{12}$$

$$t_d(f) = nf \times d \tag{13}$$

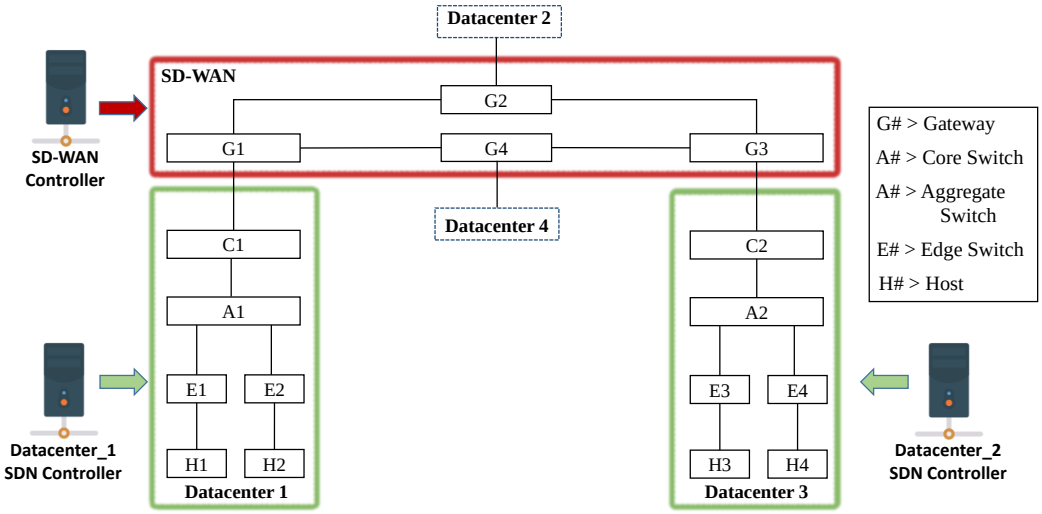


Figure 5: Multi-datacenters and SD-WAN topology for experiments.

The SD-WAN and SDN-DC are responsible for tracking the transmission time of all its applications/flows. In figure 5, 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).

$$tr(f) = \max_{n} \left(\frac{f_s}{f_{bw}} + t_d(f) \right) \tag{14}$$

The above equations are implemented in IoTSim-SDWAN . In the validation section 4, the equations are filled with numbers according to our real-world network observations using Wireshark [18] and according to TCP/IP predefined header sizes [19].

3.3. Network modeling

An SD-WAN simulator must be sufficiently flexible to support different mechanisms that allow changes in experimental contexts (e.g. network

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

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 [20] 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-DC traffic policies but also to the performance of SD-WAN and SDN-DC routing algorithms. We leverage the classical Dijkstra algorithm [21], 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 5 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 the our approach more accessible and easier to use, we have 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 15 is used to obtain the objective function of finding a route r that minimizes the traversing number of nodes in traditional networks tn

$$tn = \min_{n} \sum_{n_i, n_j \in N} r(n_i, n_j) \tag{15}$$

where (n_i, n_j) represents a link connecting two nodes.

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

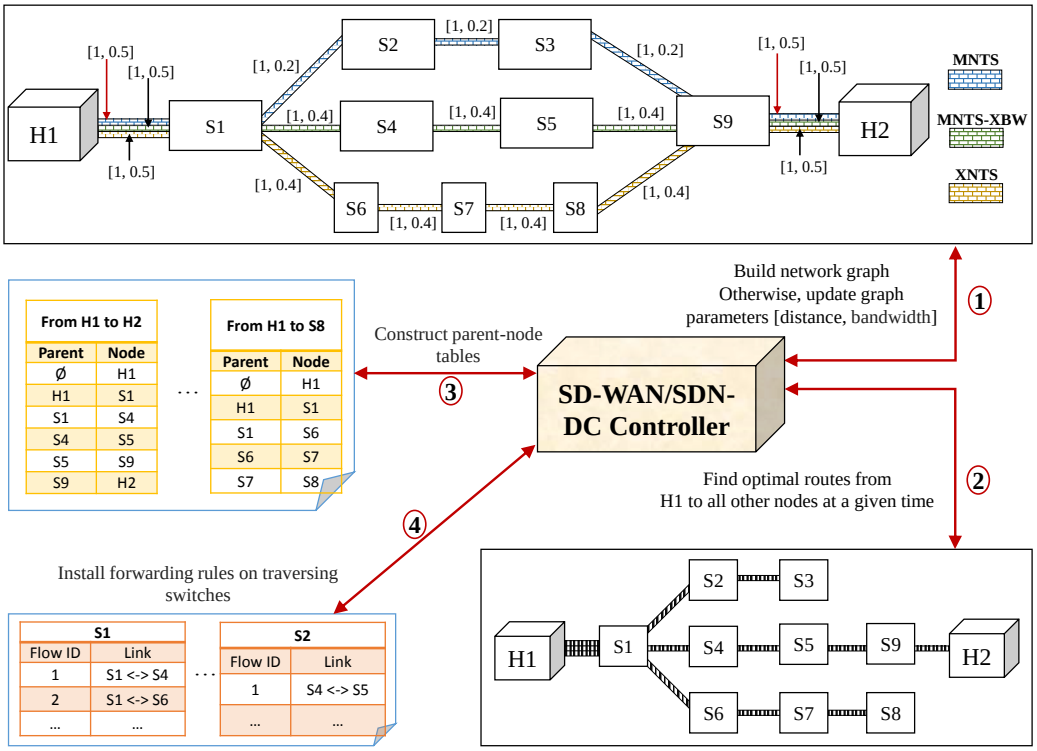


Figure 6: Overview of SD-WAN/SDN-DC controller's actions to build forwarding rules. H#: host, S#:switch, MNTS: minimum number of traversing switches, XBW: maximum bandwidth of traversing switches, XNTS: maximum number of traversing switches.

SPMB algorithm. Equation 16 is used to obtain the objective function of finding a route r that minimizes the traversing number of nodes and maximizes bandwidth bw.

$$SPMB = \max_{bw} \left(tn, \sum_{n_i, n_j \in N} r(n_i, n_j) \right) \tag{16}$$

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 6 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

Figure 7: SD-WAN Coordination Scheme

first step, every controller 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

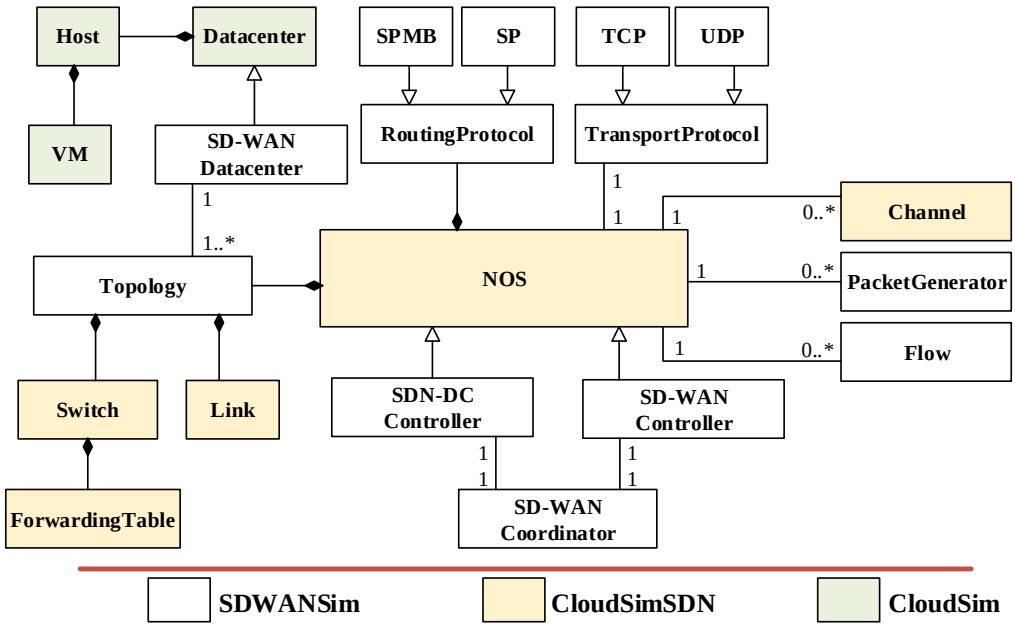


Figure 8: System structure of IoTSim-SDWAN

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 (as shown in figure 5), they must coordinate with one another in order for each controller to make appropriate (mutually agreeable) routing decisions. Figure 7 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

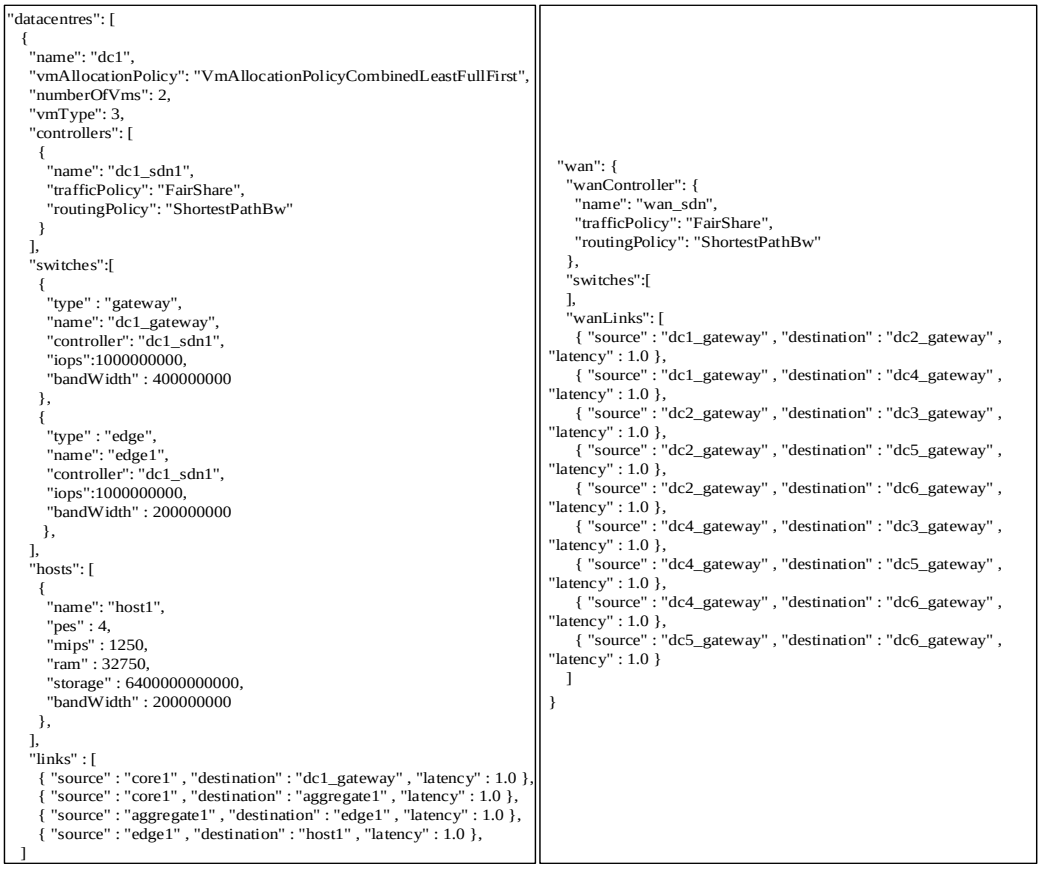


Figure 9: An example of JSON input file

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.

3.4. Components modeling and interaction of IoTSim-SDWAN

Figure 8 presents a system structure overview of IoTSim-SDWAN. As shown, IoTSim-SDWAN uses some classes of other simulating tools (CloudSim [22] and CloudSimSDN [9]). 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

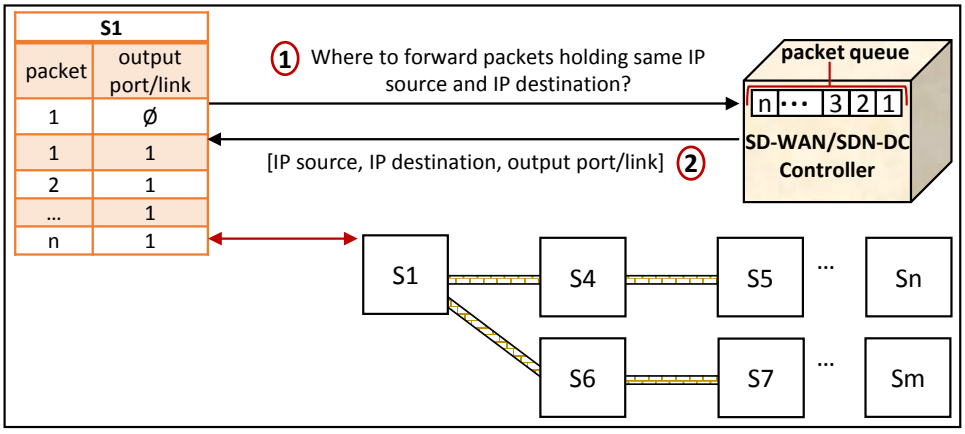


Figure 10: An interaction between OpenFlow switches (S#) and SDN for obtaining flow entry

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 9 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 3.3. 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 10 illus-

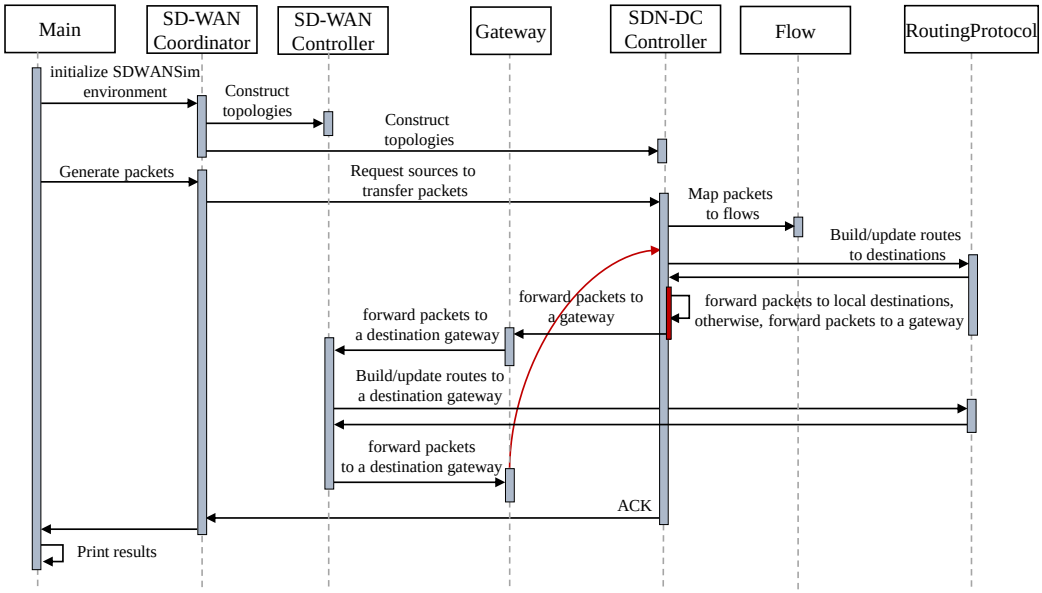


Figure 11: Underlying interactions of IoTSim-SDWAN

trates 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.

For IoTSim-SDWAN to function correctly, the transport protocol to use 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 by. In the validation section 4, 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 [18].

Figure 11 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 destina-

tion datacenter (see figure 5). 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. 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 model and reality is acceptable when measuring real network environments with their theoretical 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 effect the performance of real network systems. Based on our observations, we have derived and established the theoretical model in sub-section 3.2. The validation results proves that IoTSim-SDWAN is capable of obtaining a high degree of accuracy compared with real networks.

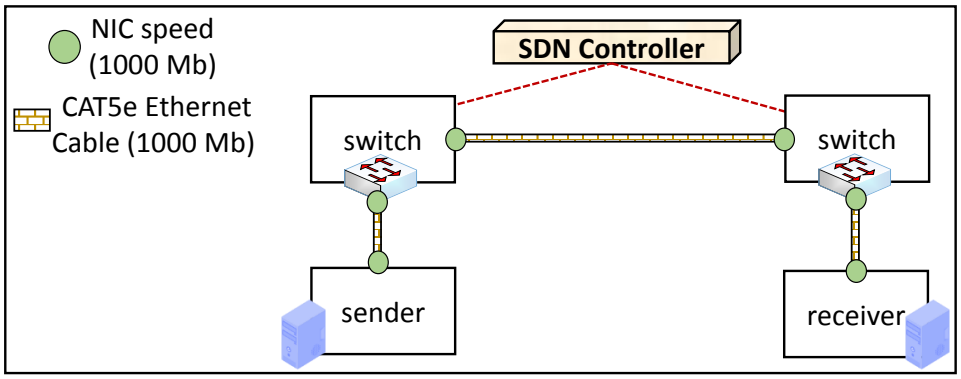


Figure 12: Network topology design for the validation experiments

4.1. 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 [23] 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) [24], which enables controllers to instruct and manipulate the data plane of switches. The SDN controller used is Ryu SDN framework [25]. 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 12 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 [26] 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 IoTSim-SDWAN.

The configuration used to validate IoTSim-SDWAN is given in Table 2. The header sizes and average payloads of given applications are obtained according to our Wireshark observations. Capturing every single delay of real network processing, queuing, transmission, and propagation is beyond the scope of this paper. 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 [27].

The management layer of the SDN is added to our empirical network

Table 2: 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

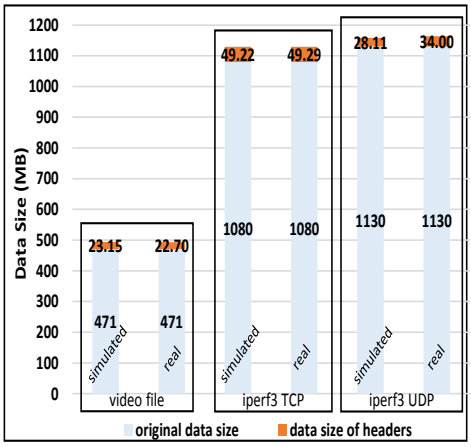
to program, manage, and instruct our network during execution (see figure 10). The SDN mechanism does not change a network's built-in capabilities, such as network bandwidth, but does manage the network in real-time (e.g. finding global optimal routes between given nodes, reserving a part of network 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 10, 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

4.2. Validation results

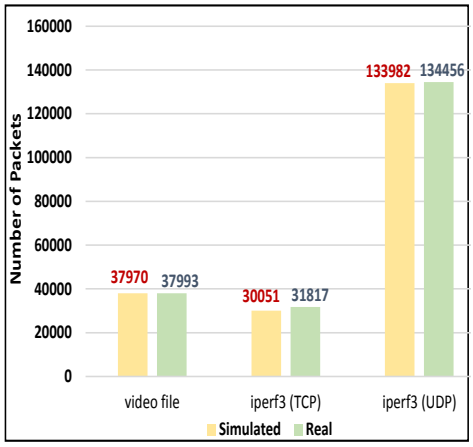
Figure 13 shows the overall header data sizes added to original data. The original data size of each application is increased when being transferred due to the added network headers. Each header size is injected by each of its respective layer as shown in the network model (see figure 3). We can see 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 13b and 13c 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 similar. The average payload factors of packets and frames as given in 3.2 verify that we can obtain similar results, even though the distribution payload sizes of packets and frames may vary in real network environments.

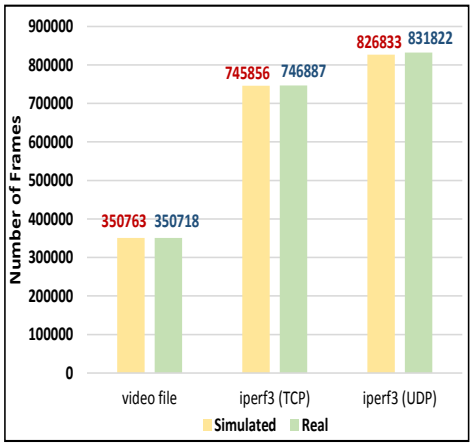
Figure 14 shows the comparison of bandwidth, speed, and delay between IoTSim-SDWAN and a real network environment. In figure 14a, the maximum bandwidth rate of Iperf3 UDP is 951Mbps while for Iperf3 TCP is 929Mbps. Iperf3 cannot achieve the speed of the theoretical network bandwidth (1000Mbps) because 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,



(a) Size of network headers added to original data



(b) Comparison of number of packets

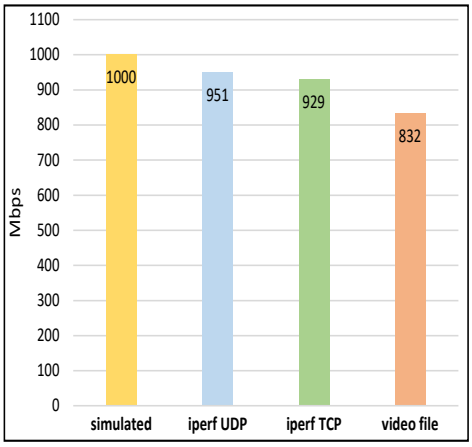


(c) Comparison of number of frames

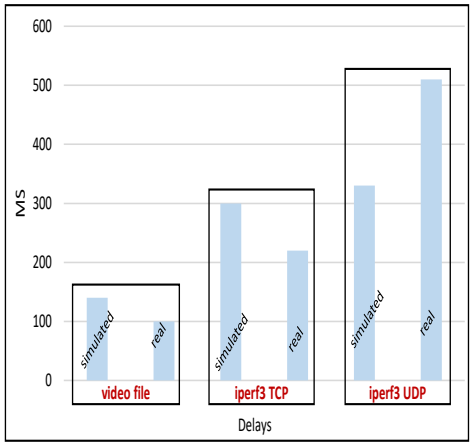
Figure 13: Comparison of IoTSim-SDWAN and a real network environment

transmitting real data (e.g., video files) not only depends on network performance but also on the performance of given hosts in terms of CPU, memory, and hard drives. Iperf3 generates random data originating from CPU (no drive) while the video file application requires data from a hard drive and then to write this data to memory, which degrades the overall performance.

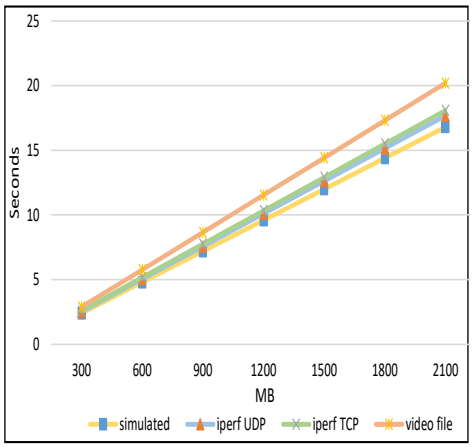
Figure 14b shows overall network delay as a result of processing, queuing, transmission, and propagation delays. We can observe that the delays



(a) Comparison of bandwidth



(b) Comparison of delays



(c) Comparison of transmission time

Figure 14: Bandwidth, speed, and delay comparison of IoTSim-SDWAN and a real network environment

of IoTSim-SDWAN and the real network are similar and, therefore, comparable. Figure 14c 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 similar. IoTSim-SDWAN and the video file has a slight difference in transmission time, which is expected since the video file depends on the

Table 3: 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

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, iperf3 is the suitable candidate to validate the accuracy of IoTSim-SDWAN.

5. 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-enabled cloud datacenters. Figure 15 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-enabled 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 3 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 2.

IoTSim-SDWAN shapes network traffic in SD-WAN and classical WAN environments according to the network model in 3.3. 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 15 conceptually illustrates how the SD-WAN and classical 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.

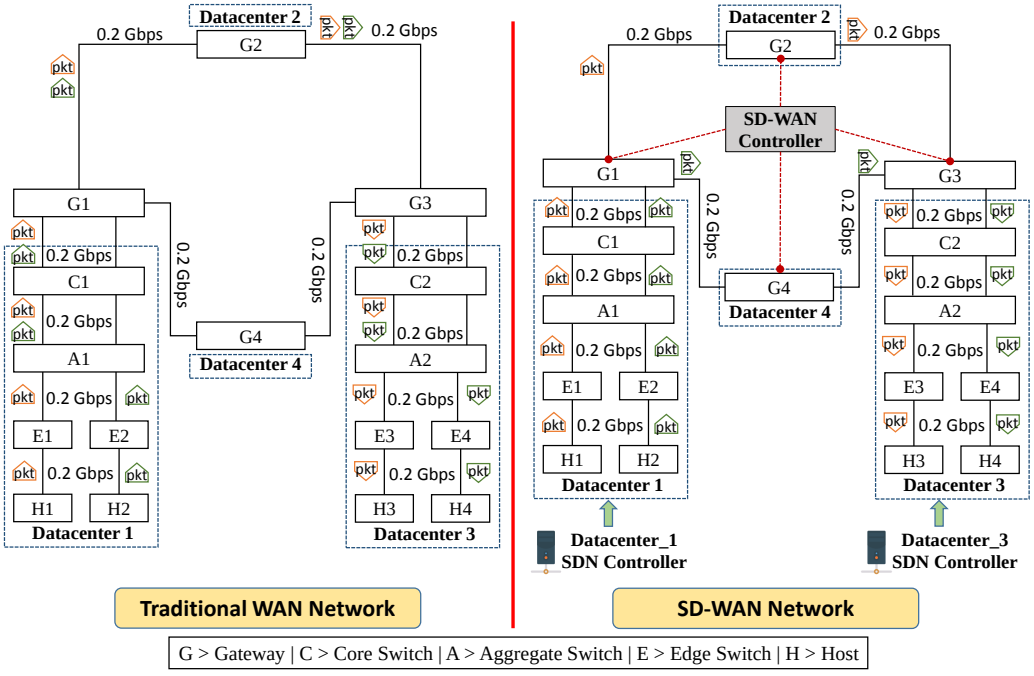
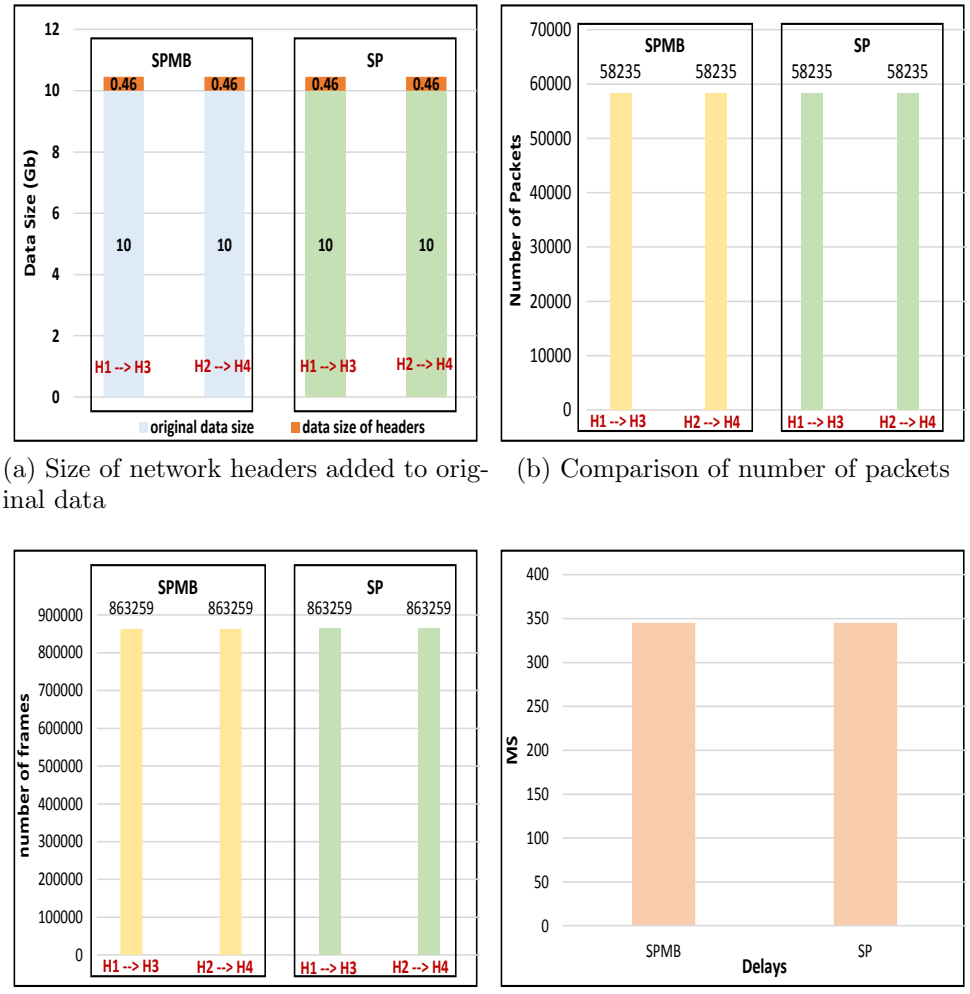


Figure 15: Network topology used in IoTSim-SDWAN

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 7.

Figure 16 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 17a illustrates the network transmission time of SD-WAN and WAN. It can be seen that SD-WAN optimized network performance and usage by decreasing the transmission time by approximately 50%. Excep-



(a) Size of network headers added to original data

(b) Comparison of number of packets

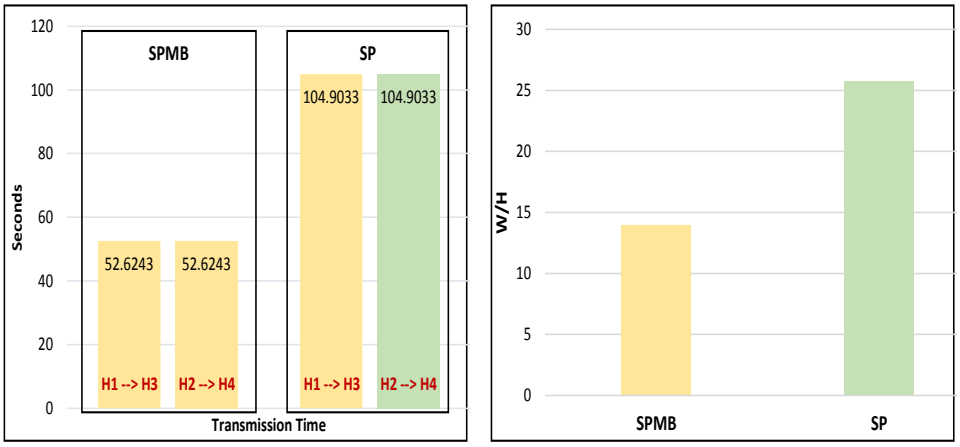
(c) Comparison of number of frames

(d) Comparison of number of frames

Figure 16: Simulation result and comparison between SPMB and SP

tional 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).

To enhance further analysis using IoTSim-SDWAN , we provide energy consumption tracking models. These allow the gathering and reporting of power consumption of switches and gateways. The modeling of power con-



(a) Data transmission time (b) Energy consumption

Figure 17: Simulation result and comparison between SPMB and SP

sumption is done by CloudSimSDN [9]. Figure 17b shows the overall energy consumption of both WAN and SD-WAN environments. As the figure shows, SD-WAN consumes less energy than WAN by approximately 54%. The reason SD-WAN consumes less energy is that packet transfer is shorter and if packets spend less time within the infrastructure then fewer resources are consumed.

6. 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) [28] 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 [22] is a discrete-event simulation tool that enables the modeling and simulation 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 [11] is a simulator designed for energy-aware cloud datacenters. It evaluates the energy consumption of datacenter components such as servers. iCanCloud [29] is a cloud-based simulator designed to conduct large-scale experiments to ease the process of integrating new policies for cloud brokers. NetworkCloudSim [12] 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 [9] is a simulation framework for SDN-enabled 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

Table 4: 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[22]	✓								
GreenCloud[11]	✓								✓
iCanCloud[29]	✓								
NetworkCloudSim[12]	✓	✓							
CloudSimSDN[9]	✓	✓		✓					✓
BigDataSDNSim[30]	✓	✓		✓			✓	✓	✓
Mininet[10]		✓		✓		✓	✓	✓	
NS-3[31]		✓				✓	✓	✓	
IoTSim-SDWAN (Proposed)	✓	✓	✓	✓	✓	✓	✓	✓	✓

the modeling and simulation of SD-WAN environments.

BigDataSDNSim [30] 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-enabled cloud datacenters. This approach also allows the simulating of any type of network topology along with providing greater flexibility for implementing new MapReduce SDN-based scheduling techniques. Nevertheless, it is limited within a single datacenter and lacks SD-WAN ecosystem properties to interconnect distributed datacenters.

Mininet [10] 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 [31] 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 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 [32] and IoTSim-Edge [33] 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. 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-enabled support both within the datacenters (local) and between them (WAN).

7. Conclusion

In this paper, we present a new tool IoTSim-SDWAN for simulating the behavior and properties of SD-WAN and SDN-enabled datacenters. IoTSim-SDWAN is a Java-based tool providing a variety of modeling approaches and functionalities to evaluate and test SD-WAN cloud-based solutions and gain additional insights on the design of future systems without requiring in-the-field experimentation that would be either prohibitively expensive or simply not practical in live systems. We model the SD-WAN ecosystem, TCP and UDP protocols, network delays, in addition to modeling the network layer of SD-WAN and classical WAN using graph theory. We propose a coordination scheme for SD-WAN and SDN controllers residing in different datacenters along with proposing appropriate routing approaches. This paper illustrates the system structure overview and physical properties of SDWANSim in addition to the interactions across IoTSim-SDWAN's constituent components.

We empirically validate and analyze the accuracy and correctness of the simulator. 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 illustrating 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.

We present a flexible approach to the design of experiments to allow researchers a seamlessly way to implement and evaluate their new SD-WAN,

SDN routing and power consumption algorithms/approaches.

The current development of IoTSim-SDWAN is limited to the network layer and does not involve the application layer of different application types (e.g. MapReduce applications). In future work we will model and implement an application layer that is distributed over multiple datacenters connected by an SD-WAN ecosystem. We also plan to add the mechanisms of edge computing and IoT, which would allow edge datacenters to interconnect with cloud datacenters via SD-WAN. In addition to this, we will propose a few novel algorithms based on our proposed IoTSim-SDWAN to accelerate the network performance of big data applications running in multiple SDN-enabled cloud datacenters.

Acknowledgement

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

References

[1] N. Abbas, M. Asim, N. Tariq, T. Baker, S. Abbas, A mechanism for securing iot-enabled applications at the fog layer, Journal of Sensor and Actuator Networks 8 (1) (2019) 16.

[2] M. Al-Khafajiy, T. Baker, A. Waraich, D. Al-Jumeily, 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.

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

[4] 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.

[5] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven wan, in: ACM SIGCOMM Computer Communication Review, Vol. 43, ACM, 2013, pp. 15–26.

[6] 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, ACM, 2013, pp. 3–14.

[7] C. Yu, J. Lan, Z. Guo, Y. Hu, T. Baker, An adaptive and lightweight update mechanism for sdn, IEEE Access 7 (2019) 12914–12927.

[8] K. Alwasel, Y. Li, P. P. Jayaraman, S. Garg, R. N. Calheiros, R. Ranjan, Programming sdn-native big data applications: Research gap analysis, IEEE Cloud Computing 4 (5) (2017) 62–71.

[9] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, R. Buyya, Cloudsimsdn: Modeling and simulation of software-defined cloud data centers, in: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2015, pp. 475–484. doi:10.1109/CCGrid.2015.87.

[10] B. Lantz, B. Heller, N. McKeown, A network in a laptop: Rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX, ACM, New York, NY, USA, 2010, pp. 19:1–19:6. doi:10.1145/1868447.1868466.
URL http://doi.acm.org/10.1145/1868447.1868466

[11] D. Kliazovich, P. Bouvry, S. U. Khan, Greencloud: a packet-level simulator of energy-aware cloud computing data centers, The Journal of Supercomputing 62 (3) (2012) 1263–1283. doi:10.1007/s11227-010-0504-1.
URL https://doi.org/10.1007/s11227-010-0504-1

[12] S. K. Garg, R. Buyya, Networkcloudsim: Modelling parallel applications in cloud simulations, in: 2011 Fourth IEEE International Conference on Utility and Cloud Computing, 2011, pp. 105–113. doi:10.1109/UCC.2011.24.

[13] aryaka, https://www.aryaka.com/sd-wan-as-a-service/, accessed October 30, 2019.

[14] silver-peak, https://www.silver-peak.com/, accessed October 30, 2019.

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

[16] T. Kelly, Scalable tcp: Improving performance in highspeed wide area networks, ACM SIGCOMM computer communication Review 33 (2) (2003) 83–91.

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

[18] Wireshark, https://www.wireshark.org/, accessed July 22, 2019.

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

[20] The internet topology zoo, http://www.topology-zoo.org, accessed July 24, 2019.

[21] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische mathematik 1 (1) (1959) 269–271.

[22] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. Pract. Exper. 41 (1) (2011) 23–50. doi:10.1002/spe.995.
URL http://dx.doi.org/10.1002/spe.995

[23] Shenzhen helor cloud computer, http://www.hlypc.com/, accessed July 24, 2019.

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

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

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

[27] R. Ramaswamy, N. Weng, T. Wolf, Characterizing network processing delay, in: IEEE Global Telecommunications Conference, 2004. GLOBECOM'04., Vol. 3, IEEE, 2004, pp. 1629–1634.

[28] E. Rosen, A. Viswanathan, R. Callon, Multiprotocol label switching architecture, Tech. rep. (2000).

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

[30] K. Alwasel, R. N. Calheiros, S. Garg, R. Buyya, R. Ranjan, Bigdatasdnsim: A simulator for analyzing big data applications in software-defined cloud data centers, arXiv preprint arXiv:1910.04517.

[31] G. F. Riley, T. R. Henderson, The ns-3 Network Simulator, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 15–34. doi:10.1007/978-3-642-12331-3_2.
URL https://doi.org/10.1007/978-3-642-12331-3_2

[32] C. Sonmez, A. Ozgovde, C. Ersoy, Edgecloudsim: An environment for performance evaluation of edge computing systems, Transactions on Emerging Telecommunications Technologies 29 (11) (2018) e3493.

[33] 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.