Vrije Universiteit Amsterdam

Honours Programme, Project Report

Exploring Energy Efficiency and QoS Impact of Network Design in Datacenters Through Large-Scale Simulation

---

Author: Alessio Leonardo Tomei (2777731)

1st supervisor:	Alexandru Iosup
daily supervisor, if different:	Jesse Donkervliet and Dante Niewenhuis
2nd reader:	Tiziano de Matteis

A report submitted in fulfillment of the requirements for the Honours Programme, which is an excellence annotation to the VU Bachelor of Science degree in Computer Science/Artificial Intelligence/Information Sciences
version 1.0

April 14, 2025

Abstract

Our society is rapidly becoming more digital, with online activities driving a significant portion of this transformation. To support this massive increase in online traffic, particularly in sectors like online gaming, video streaming, and e-commerce, datacenters have become indispensable. Simulation is a critical part of exploring datacenter technologies, enabling scalable experimentation and what-if analysis in a matter of minutes to hours. Regrettably, the networking component is frequently neglected, despite accounting for nearly 20% of the total energy consumption in datacenters and being essential for ensuring high *quality of service* (QoS) delivery. This highlights the need for simulators that not only incorporate emerging technologies and applications but also provide comprehensive modeling of networking interactions within the datacenter environment. Addressing this necessity, we enhance the capabilities of OpenDC, a state-of-the-art datacenter simulator, which already supports features such as machine learning, *function-as-a-service* (FaaS), serverless computing and the procurement of HCP-as-a-service infrastructure. By introducing networking simulation capabilities, we position OpenDC as a pioneering tool for simulating compute-network interactions and in-depth analysis of datacenters energy consumption and QoS. Our design supports both networking-only and compute-networking workloads, including a comprehensive library of prefabricated components and topologies for constructing and sharing datacenter designs, and accommodating a wide range of input formats and output metrics. We also focused on convenience and extensibility, with an interactive interface aimed at educational use and ease of adoption, and adhering to modern software development standards. The ability to scale from small-scale experiments using an interactive interface to large-scale analyses of energy consumption and QoS, with its integration with compute simulations, positions OpenDC as a valuable tool for a wide range of users, including students, researchers, and datacenter operators.

Contents

1 Introduction

In this section we discuss the context of the work, diving into the basics of cloud datacenters, introducing research questions and contributions of this paper.

1.1 Context

The strain on network infrastructures is escalating due to a rapid increase in internet traffic, driven by factors such as the rise of high-definition content, the proliferation of connected devices, and the growth of data-intensive applications. Each day, over 400 exabytes of data are generated, and in 2025, this number is projected to surge to almost 500 exabytes per day (≈23% increase) [44, 3]. Nokia's *Global Network Traffic Report* [2] predicts that global network traffic demand will reach between 2,443 to 3,109 exabytes per month in 2030, with a *compounded annual growth rate* (CAGR) reaching as high as 32%. One relevant factor of the increasing traffic demand is online gaming, which ranks as one of the fastest growing sectors in the entertainment industry [42] and in particular of cloud gaming, which relies on remote servers for running and rendering games, streaming in real time the player *point of view (POV)*. Depending on network and video quality, cloud gaming can consume up to 45 Mb/s [16]. To support this massive increase in online traffic, particularly in sectors like online gaming, video streaming, and e-commerce, datacenters have become indispensable, being central to the operations of virtually every online service. As the demand for low-latency gaming experiences and high-quality video streams increases, so does the need for scalable datacenters capable of delivering high QoS. As these facilities expand to meet rising data demands, their substantial energy consumption and carbon footprint have come under scrutiny. This has led to increased focus on assessing their environmental impact.

1.2 Problem

The global energy crisis has become a pressing issue, exacerbated by the growing demand for energy across various sectors and the slow transition to renewable sources. Current energy infrastructures are struggling to meet the increasing consumption rates, leading to concerns about energy security, and environmental sustainability [15]. According to recent studies, the strain on energy resources is particularly evident in sectors like *Information and Communication Technology* (ICT), where datacenters alone are projected to account for about 4.5% of the global electricity demand in 2025 [33]. Despite accounting for about 20% of the total datacenter energy consumption [14], datacenter's networking is rarely the research focus in the field. The alarming energy impact of datacenter has brought the need to analyze their energy consumption to find ways to reduce their energy footprint without compromising QoS. One approach to evaluating datacenter performance is benchmarking, which involves monitoring and recording various metrics over a defined time period. However, this method presents significant challenges, as it requires substantial investments of both time and capital. Additionally, it is advantageous to predict the performance and energy impact of a given configuration prior to its deployment, minimizing the risk of suboptimal performance and ensuring that the system meets operational expectations.

1.3 Research Questions

RQ1 How to model network resources in datacenters?

This research revolves around the effective modeling of network resources within datacenters (Section 3), and how to accurately predict their usage and their impact on QoS and energy consumption. Network is a critical component in datacenters, however there are no existing standard methodologies or models for large scale datacenter networks.

RQ2 How to integrate network and compute workloads in an event-driven datacenter simulation?

It is important to enable comprehensive evaluation of datacenter infrastructures, combining compute and network simulation. The main challenge derives from the complexity of integrating the network design into existing systems and simulators, while also providing the flexibility of network-only simulation, experimentation and evaluation (Section 4).

RQ3 How do datacenter networks affect their energy use and QoS?

Understanding the impact of datacenter networks have on metrics such as energy consumption and QoS is essential. We aim to demonstrate the effectiveness of our simulator through a series of experiments designed to validate its functionality, while gaining insights into the impact of isolated components and policies on global datacenter energy consumption and QoS (Section 5). Additionally, we evaluate the worst-case simulator performance and scalability across different independent variables, providing a clearer understanding of the simulator's scalability and potential areas for improvement (Section 5.5).

1.4 Main Contributions

To address the research questions outlined in Section 1.3, we present the following main contributions:

MC1 We present a state-of-the-art, event-driven datacenter network simulator, designed to support the modeling and simulation of arbitrary network topologies through an intuitive *command-line interface (CLI)* (Section 3).

The simulator, developed following modern engineering practices, offers advanced features, including support for various energy models, QoS policies, and *software-defined networking* (SDN). Additionally, it facilitates the easy sharing and reuse of topologies and scenarios, providing a flexible and comprehensive tool for network simulation.

MC2 We integrate the network simulator with OpenDC, a highly regarded datacenter simulator developed over the course of 7+ years [36, 28, 37] (Section 4).

This integration results in a robust tool capable of simulating comprehensive datacenter scenarios, including both computational and networking workloads. By accurately modeling the interactions between these components, OpenDC provides a more holistic evaluation of datacenter performance compared to other simulators.

MC3 We present some use cases for datacenter simulation utilizing the enhanced version of OpenDC (Section 5).

These use cases include educational and interactive simulations, energy consumption analysis across different network topologies and policies, *quality of service* (QoS) and congestion analysis, as well as validation and reproducibility of simulation results. Together, these scenarios demonstrate the versatility and applicability of the simulation framework in addressing diverse research challenges. Additionally, we gain new insights, e.g. [add result examples].

2 Background

In this section, we aim to provide an overview of the subjects necessary to comprehend the rest of the paper. Section 2.1 examines the potential consequences of network infrastructure scalability limitations during periods of increased demand, exemplified by the recent COVID-19 pandemic. Section 2.2 offers a brief explanation of the concept of datacenter, while Section 2.3 presents an overview of a widely adopted DCN architecture, with a focus on online gaming. Sections 2.4 and 2.5 discuss various network topologies and policies, highlighting their significant impact on performance. In Section 2.6 we introduce the principles of DCN energy models, accompanied by a practical example. Finally, Section 2.7 outlines the DCN performance metrics that are referenced throughout the paper.

2.1 Network Traffic Demand During and After the Pandemic

During the COVID-19 pandemic, global network infrastructures faced an unprecedented surge in traffic demand, driven by the rapid transition to remote work, online education, and increased reliance on digital services such as video conferencing, streaming platforms and cloud based applications. In fact, network traffic volume increased by approximately 15-20% within just one week following the pandemic breakout, pushing network capacity to its limits [18]. As outlined in the problem section (1.2), although the pandemic has subsided, network traffic volumes are expected to continue rising. During this period, many datacenters struggled to handle the spike in traffic, leading to performance bottlenecks and service degradation. This unprecedented demand underscored the need for more robust, flexible, and scalable network architectures capable of handling sudden spikes in traffic. The pandemic increased the focus on upgrading infrastructure with advanced networking solutions such as *software-defined networking* (SDN), and *network function virtualization* (NFV), which offer greater adaptability and resource efficiency. One of the key contributions of our research is to facilitate the advancement of datacenter networking technologies by providing tools for simulating and evaluating modern datacenter networking concepts, including SDN and NFV. Our tools allow researchers and practitioners to explore the performance, scalability, and efficiency of various datacenter networking solutions guiding the development and optimization of next-generation datacenter infrastructures, ensuring they can meet the increasing demands of contemporary cloud-based services.

2.2 What is a Datacenter?

Datacenters are physical facilities housing vast networks of interconnected servers and storage systems, along with critical infrastructure for power distribution, cooling, and high-speed internet access. In the past, organizations typically managed their IT infrastructure on-premise or rented space in co-located facilities. Today, however, most digital services are deployed in the cloud, where cloud providers handle the hardware and operational lifecycle, often using hyperscale datacenters.

2.3 Online Gaming Perspective of Legacy Datacenter Design

In this section, we provide an overview of datacenter network design and operations, focusing on a widely adopted legacy 3-tier topology. While the analysis is framed from

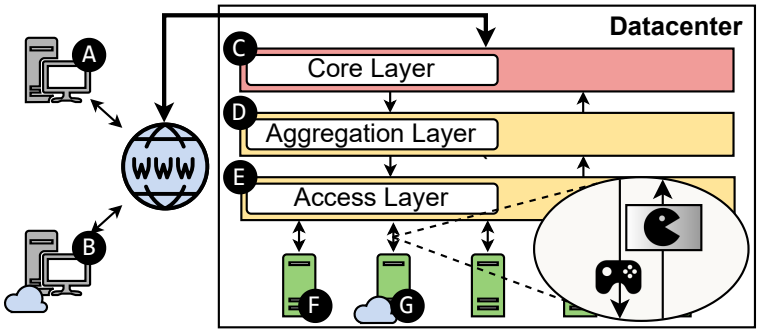


Figure 1: A system model for Online Gaming in Datacenters.

the perspective of online gaming, the principles discussed are broadly applicable to a variety of cloud services.

Online gaming implies huge amount of data transmitted over the Internet. Data can represent player actions, updates for entities and NPCs (non-player character), and, in the case of games with *modifiable virtual environment* (MVE), also modifications of the world. The amount of data varies widely depending on genre, such as *first-person shooters* (FPSs), *real time strategy* (RTSs), *massive multiplayer online* (MMOs), MVEs, etc. In Figure 1, we present a model of online gaming in a three-tier datacenter architecture, which has been the standard model for decades and still used in numerous datacenters. In particular, the figure highlights the different hops that game packets need to traverse inside a datacenter to reach either the server (client-to-server communication) or the inter-datacenter network (server-to-client communication).

The model starts with *players* (or *users*) (Ⓐ), which are connected to the Internet and exchange packets with the server (Ⓕ). Packets are redirected and distributed by many switches on different tree layers. The Core Layer (Ⓒ) is the top of the hierarchy, provides the interconnection of multiple datacenter modules and handles ingoing and outgoing traffic, normally with high performance 10GE switches [25]. The Aggregation Layer (Ⓓ) provides and manages many functions and services such as firewalls, as well as connecting core and access layers. The Access Layer (Ⓔ) consists of top of rack (ToR) switches, which are responsible for rerouting packets directly to the different servers deployed. Each ToR usually connects 10-20 different servers with 1GE ports [25, 40]. This datacenter network perspective is particularly important considering the large majority of datacenter network energy expenditure is due to switches [26].

An additional factor to be considered is the rise in popularity of cloud gaming, which relies on remote servers for running and rendering games, streaming in real time the player *point of view* (POV). Depending on network and video quality, cloud gaming can consume up to 45 Mb/s [16]. In such a scenario, the game is run in the server (Ⓖ), and a larger amount of data is transmitted from the server to end-point user (Ⓒ) in the form of a stream of *frames*, rendered in response to the user inputs that are sent to the server. Other deployments are possible, for example rendering could be delegated to *thin clients*,

closer to the user, reducing latency and network traffic [27].

2.4 Diverse and Emerging Topologies

Datacenter network topologies can be broadly classified into three primary categories: *tree-based*, *recursive* and *direct networks*. Tree-based topologies such as VL2 [22] and 2 tier *leaf-spine* topology [8]) are widely used in practice and structured hierarchically, with the servers as leaf nodes and switches as intermediate nodes. *Fat-Tree*, introduced by *Al-Fares et al.* [7], is a variation of the classical tree structure that features a multi-rooted design for improved fault tolerance and bandwidth. Recursive-defined topologies such as *DCell* [24], *BCube* [23], *FinConn* [31] and *FlatNet* [32], can be regarded as partially server-centric topologies, as servers also function as routing nodes within the network. In this configuration, servers play a dual role, handling both computational tasks and network traffic routing. However, servers may become themselves bottlenecks, and they cannot be powered off, since they need to remain operational to maintain network connectivity. Direct Network topologies such as *CamCube* [5] and *NovaCube* [51] establish direct connections between servers, enabling applications to implement custom routing protocols for improved performance. These topologies offer several advantages over switch-centric designs, including enhanced reliability, reduced deployment costs, and lower energy consumption. However, they also present challenges, particularly in the form of longer average routing paths, which can lead to increased latency, and less favorable worst-case performance scenarios.

A datacenter network simulator capable of building and simulating arbitrary topologies is vital for enabling researchers and practitioners to conduct rigorous evaluation and optimization of both established and emerging architectures.

2.5 Diverse Routing Protocols

For decades, internet routing protocols have leveraged shortest-path algorithms to optimize data transmission, such as the *Routing Internet Protocol* (RIP), *Open Shortest Path First* (OSPF) [53], *Intermediate System to Intermediate System* (IS-IS), *and Border Gateway Protocol* (BGP) being widely used. In theory, routing in datacenter networks can be simplified by utilizing predetermined lookup tables. Such an approach significantly reduces the computational complexity of routing decisions. However, datacenter networks are prone to frequent node and link failures, as well as congestion [12, 21], requiring more adaptive routing mechanisms. In modern datacenter networks, shortest-path algorithms are also being used, especially when combined with load balancing [48]. Load balancing promotes even utilization of network links and switch buffers. *Equal Cost Multi-Path* ECMP [29], which distributes network flows across equal-cost paths, has emerged as a popular choice in datacenter, reducing congestions [9, 10, 34]. However, variations in flow sizes can lead to disproportionate link utilization, reducing this approach effectiveness. Network operators frequently design custom control planes to manage load balancing and routing in software, known as *Software-Defined Networking* (SDN) [17]. In SDN architectures, the control plane is decoupled from the data plane and is implemented through SDN controllers, which operate independently of the physical switching devices. While

centralized SDN offers flexibility and centralized control, distributed routing protocols are generally more suitable for large-scale datacenter networks due to their scalability and ability to quickly adapt to node or link failures. However, the reliance on a centralized SDN controller can limit both network size and overall reliability [49]. In distributed routing protocols, the routing tasks are executed directly on the switching nodes, enabling parallelized operations that support more extensive and resilient networks.

A datacenter network simulator must possess the capability to simulate arbitrary routing protocols, whether they are SDN-based, centralized or distributed. The flexibility to model various routing strategies is essential for evaluating their performance under different conditions, such as node failures, link congestion, and traffic variations, allowing researchers and network operators to explore trade-offs between affordability, scalability and fault tolerance.

2.6 Network Energy Models and policies

Abts et al. showed that the datacenter network component can account for a substantial portion of overall cluster IT energy usage, potentially reaching up to 50% [4]. Numerous energy models have been proposed to model datacenter network energy consumption. One example of such model is the one proposed by *Wang et al.* [52]. The paper models network energy consumption around switches (switch-centric), here we present a simplified version that does not take into account *consolidation periods*.

P^{switch}: power draw of a switch

$P^{chassis}$: power draw of the chassis of a switch

P^{cards}: power draw of the line cards of a switch

P^{port_d}: dynamic power draw of a port

P^{port_s}: static power draw of a port

rx: receiving data rate on a port

tx: transmitting data rate on a port

bw^{link}: the total bandwidth of a link

$$\sum_{k=1}^{N} P_k^{switch} \tag{1}$$

$$P_k^{switch} = P_i^{chassis} + P_i^{cards} + \sum_{j=1}^{p} (P_{i,j}^{port_s} + P_{i,j}^{port_d}) \tag{2}$$

$$P_{i,j}^{port_d} = P_{i,j}^{port_f} * \frac{rx_{i,j} + tx_{i,j}}{bw_{i,j}^{link}} \tag{3}$$

While numerous energy models have been proposed for datacenter networks, there is consensus that switching hardware significantly impacts overall energy consumption. However, the actual utilization of these switches has a minimal effect, typically only 5 to 10% of the total power draw [52, 26]. These observations have driven considerable efforts towards the development of energy-proportional datacenter networks [4], whose goal is to ensure energy consumption scales linearly with network utilization. However, datacenter traffic generally operates well below peak capacity, leading to considerable energy waste. Significant has been conducted in formulating *energy-aware routing* SDN strategies to reduce energy consumption in datacenter networks [26, 45]. These strategies, by selectively utilizing the minimum number of devices required for routing, without degrading network performance, and transitioning idle devices into sleep or shutdown modes, can significantly lower power consumption.

A datacenter network simulator capable of modeling SDN-based energy-aware policies is essential for enabling researchers and practitioners to conduct comprehensive evaluations and optimizations of both existing and emerging policies.

2.7 Network Metrics

In this section, we present the primary network performance metrics that will be used throughout the remainder of the paper.

2.7.1 Network Power Usage Effectiveness (NPUE)

A variety of metrics are currently employed to evaluate datacenters, but one particular metric has emerged as a predominant industry standard. The *Power Usage Effectiveness* (PUE) metric, introduced in 2006 [35], has become the most widely utilized measure for assessing the energy efficiency of datacenters [13, 30]. The PUE, defined in Eq 4, quantifies the proportion of energy utilized specifically by IT equipment relative to the total energy consumption of a datacenter.

$$PUE = \frac{total\_facility\_energy\_use}{IT\_energy\_use} \tag{4}$$

In this work we focus on a variation of PUE, namely *Network Power Usage Effectiveness* (NPUE), introduced by *Popoola et al.* [41]. This metric represents the ratio of overall IT power to power utilized by the network modules, defined in Eq 5.

$$NPUE = \frac{IT\_energy\_use}{network\_energy\_use} \tag{5}$$

2.7.2 Network Power Effectiveness (NPE)

Despite very similar names, *Network Power Effectiveness* (NPE) and NPUE are very different metrics. NPE, introduced by *Shang et al.* in 1025 [46], is defined as the ratio

between the aggregate network throughput and the total network power consumption, as described in Eq 6. This metric quantifies the end-to-end bits per second per watt (or equivalently, bits per joule) during data transmission. Consequently, NPE reflects the tradeoff between power consumption and network throughput in datacenters.

$$NPE = \frac{aggregate\_throughput}{network\_power\_use} \tag{6}$$

2.7.3 Communication Network Energy Efficiency (CNEE)

Communication Network Energy Efficiency (CNEE), described in Eq 7, quantifies the efficiency with which the network converts electrical energy into information transmission [20]. This metric measures how effectively the network performs the task of data delivery.

$$CNEE = \frac{network\_power\_use}{aggregate\_throughput} \tag{7}$$

3 Design of a Discrete-Event-Event Datacenter Network Simulator (RQ1)

One of the key contributions of this research is the development of a tool capable not only of networking simulation but also of more comprehensive scenarios, incorporating computational and networking performance analysis. In this section, we will design a networking simulator, which can optionally be included in a datacenter simulator (Section 4.6).

3.1 Requirements

In this section, we determine requirements that our datacenter network simulator should address.

R1 Combine compute and network simulations in a single instrument, to allow researchers and other users to explore the impact of networking infrastructures in datacenters. This combined approach facilitates a more comprehensive evaluation but introduces significant challenges due to the required compatibility with potentially diverse compute datacenter simulators (an example is provided in Section 4.6).

R2 Support the most well-known as well as user-defined topologies. To facilitate research and experimentation in the field, as well as educational purposes, the system should be designed to be generic, capable of supporting any network topology, through detailed user-defined components, while providing shortcuts for constructing commonly used and recursively defined, as well as switch-centric and server-centric topologies. While this flexibility enables a broad range of experimentation, it also introduces significant design complexity.

R3 Enable reuse and sharing of designs of topologies and single components. To simplify the complex process of datacenter network design, and minimize the barrier to entry, the system should support reuse and sharing of network designs and single components. This feature also facilitates *repeatability* and *reproducibility* of results [1], allowing experiments to be conducted with the same shared components and scenarios. However, this introduces the challenge of defining a non-volatile representation for a wide range of network components, ensuring that they can be easily loaded, exported and combined by the simulator.

R4 Support easy addition and modification of policies and protocols. The system should offer pre-defined routing protocols, as well as QoS and energy-saving policies, reducing the barrier to entry. These policies should be applicable on the network, node and job level, to enable *Software Defined Networking* (SDN) and network aware strategies. Additionally, the system must provide users with the flexibility to design and test custom policies, with minimal effort, and easily support their testing. In particular, the simulator should support multipath routing and intra-datacenter networking, features often missing on other simulators. However, with such a generic system that allows users to define their own policies, the complexity of the system increases. This complexity is further compounded by the need to provide users with tools for effectively testing and debugging these custom algorithms and policies.

R5 Support educational purposes with interactive simulation interface. The system should offer tools designed for training and educational purposes, featuring an interactive interface that facilitates guided experiments on a small scale. This introduces a fundamentally different mode of interaction with the simulator, necessitating additional features and expanding the scope of the overall project.

R6 Permissive trace format, to account for missing values and support the wide range of (non-standardized) traces that are publicly available. However, this permissive format adds complexity to the system, as it must be capable of converting diverse formats into valid network workloads that can be interpreted and executed by the simulator.

R7 Provide comprehensive output metrics with adjustable granularity. The system should offer a comprehensive range of output metrics, covering both QoS and energy consumption. Users should have the ability to selectively choose the relevant metrics to be exported as well as the output granularity, minimizing unnecessary boilerplate output and reducing the size of output files. However, frequently monitoring a wide range of parameters through recurring snapshots could degrade the simulator's performance, so measures must be taken to prevent this.

R8 Adhere to modern software development standards. The system should be designed to evolve and adapt to future research and extensions. Therefore, it must be developed according to modern, professional engineering standards. [EXPAND/DEFINE]

R9 Ensure high performance and scalability of the simulator. The system must handle large-scale simulations with limited performance overhead. In particular, one should be able to simulate millions of network events (Section 3.5) across hundreds of nodes within seconds or a few minutes. This presents a significant challenge due to the substantial computational and memory demands associated with simulation of large-scale and complex networks.

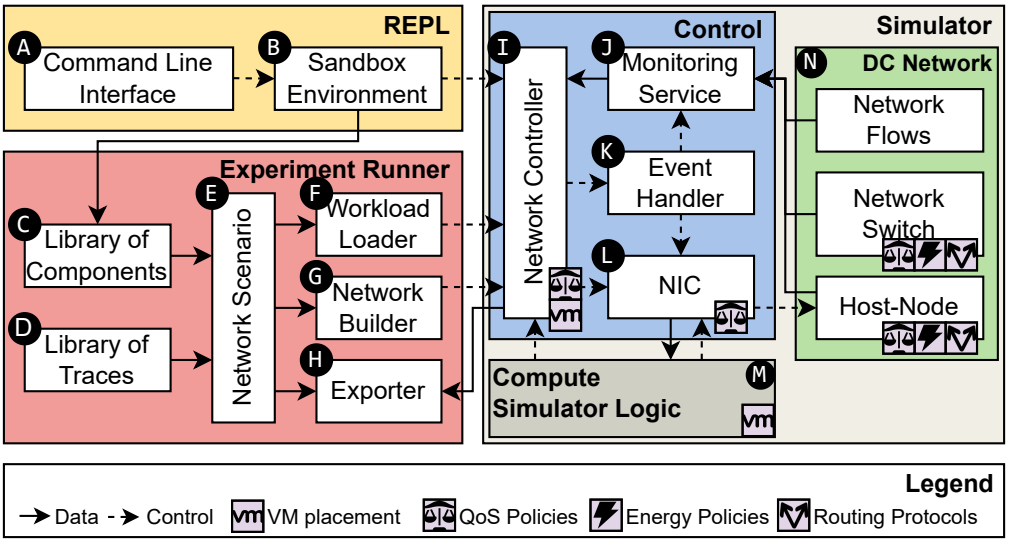


Figure 2: An overview of the architecture of the DC Network Simulator.

3.2 Design Overview

In this Section, we discuss the high-level architecture of the simulator, as shown in Figure 2. At the highest level, we identify four main components: an experiment runner for workload-based experiments; the REPL environment, which facilitates experimentation, the creation of new pre-fabricated components, as well as testing and debugging; a set of structures responsible for controlling the network's discrete event simulation and providing snapshots of the system's state; and, finally, the datacenter network model itself. Additionally, the figure illustrates, at a high level, how a compute simulator module could interact with our simulator. The policy management aspects will be discussed in Section 3.4. We now proceed to discuss each of these components and their respective subcomponents.

REPL (addresses requirements R4, R5): The *Command Line Interface* (Ⓐ) is beneficial for users as it allows efficiently set up, configure, and reproduce experiments. It is also through the CLI that users can benefit from the *REPL Environment*. The *Sandbox Environment* (Ⓑ) (Section 4.5) provides an interactive environment for event-driven simulations as well as for building and modifying components and topologies. It allows users to rapidly test newly defined policies and protocols in a more debugging-friendly environment, addressing **R4**. This feature is particularly valuable for educational purposes and facilitates plug-and-play simulations, exploring network behaviors with ease, addressing **R5**.

Experiment Runner (addresses requirements R2, R3, R6, R7): The *Library of Components* (Ⓒ) offers predefined network components and topologies, simplifying the design process for researchers and practitioners. The library enables construction

of complex datacenter networks without the need to design each element from scratch, minimizing the barrier to entry. The library includes commonly deployed architectures (e.g., Fat-Tree, Spine-Leaf). Users can extend the library by incorporating their own custom-defined components and topologies, promoting their reuse and sharing, addressing **R3**. The *Library of Traces* (D) provides a set of predefined network traces representing a wide range of network workloads (Section 4.3.2). This feature enables researchers and datacenter operators to initiate experiments without the time-consuming task of generating custom network traces. While using predefined traces provides a convenient starting point, producing specific traces may ultimately yield more representative results for specific environments. The *Network Scenario* (E) serves as an aggregator that encapsulates all the necessary information required for a given simulation, including monitor granularity and selection of output metrics, partly addressing **R7**. The network scenario functions as the final input format for the simulator (Section 4.3.3). The *Workload Loader* (F) converts a wide range of publicly available input traces into a standardized workload executable by the simulator (Section 4.3.2), addressing **R6**. The *Network Builder* (G) converts network components and topologies, that may be defined recursively, or manually, from raw JSON files to their runtime representation, partly addressing **R2** The *Exporter* (H) (Section 4.4) handles exporting tracked metrics to output files, according to configurations provided by the network scenario. It allows to selectively choose which metrics are to be exported among the many available and with which granularity, enhancing the control the user has over the output.

Control (addresses requirement R1, R7, R9): The network logic is controlled by this module, which handles the simulation and the monitoring of resources, while ensuing scalability and minimal performance degradation, addressing **R9**. The *Network Controller* (I) serves as the interface that triggers every event within the network, including connection and disconnection of components, while supplying real-time monitoring data to higher-level components. This component also supports integration with an external compute simulator, delegating control over network behavior and providing real time feedback for possible workload dependencies between the two modules, addressing **R1**. The *Monitoring Service* (J) provides real-time monitoring of a wide range of network QoS and energy consumption metrics (Section 4.4, at the level of individual flows, nodes, and the entire network, addressing **R7**. The *Event Handler* (K) handles network events (Section 3.5, ensuring their order of execution and consistency with the simulation virtual time (Section 3.5). The *Network Interface Controller* (L) serves as an abstraction provided by certain network nodes that enables to directly manage their network behavior, while providing real-time information about network performances, allowing adjustment in response to changing conditions. It is especially used in combined compute-network simulations, addressing **R1**.

Compute Simulator Logic (addresses requirement R8): Our design facilitates not only independent network simulations but also the integration of networking and computing simulations. This capability is illustrated in the figure by the *Compute Simulator Logic* (P), which can manage network operations either through the Network Controller or via NICs. This approach enables a detailed representation of the interactions between networking and computing components.

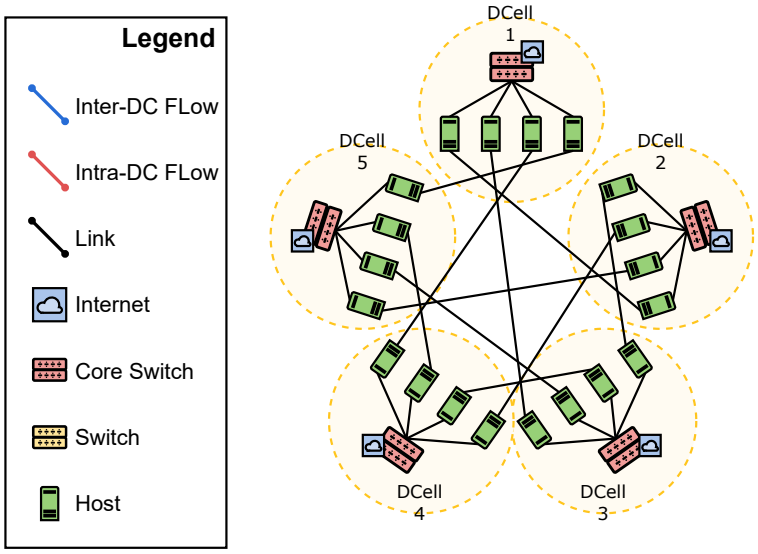


Figure 3: DCell-based topology.

DC Network (addresses requirements R4, R8): The *DC Network* Q (Section 3.3) is built of all different network components (e.g. switches, host-nodes, links), and accurately simulates their interactions. These interactions are modeled in a highly generic manner, enabling the representation of virtually any network topology, provided it is manually specified, addressing **R2**.

3.3 Modeling Network Resources

In this Section, we present our network model and its key components by examining two distinct example topologies: Fat-Tree [7], illustrated in Figure 4, and DCell [24], shown in Figure 3. These topologies are fundamentally different, with the Fat-Tree being switch-centric, meaning routing is handled exclusively by switches, while DCell exemplifies a server-centric (recursively defined) architecture, where routing is managed by both servers and switches. These examples demonstrate the simulator's versatility in modeling various topologies. We begin by outlining the primary characteristics of these topologies before delving into our network model, referencing them throughout.

DCell: Figure 3 shows a DCell, which is an example of a recursive defined topology. It's most basic element DCell_0 is built with n servers connected to an n-port switch. It is a recursive-defined topology since you can build DCell_k connecting multiple DCell_{k-1}.

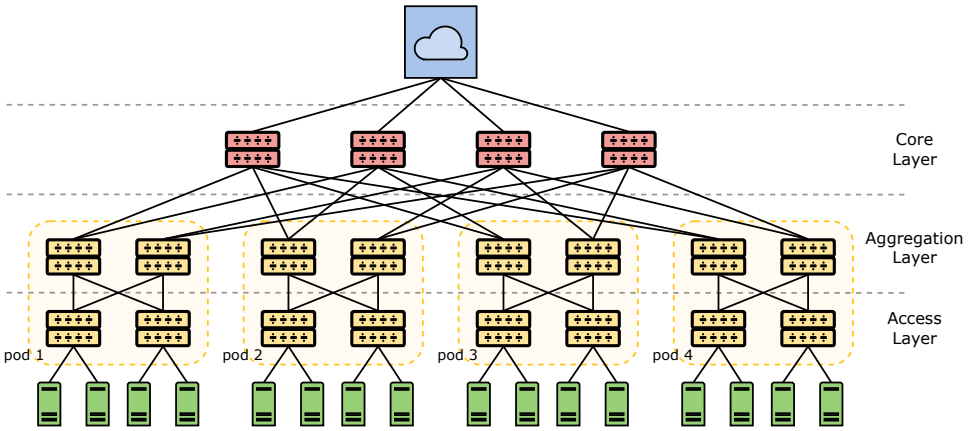


Figure 4: FatTree topology with n=4.

Fat-Tree: Fat-Tree, shown in Figure 4 is a widely used switch-centric topology, also used by researchers in projects such as Portland [39], Hedera [6], ElasticTree [26], etc. Fat Tree is a multi-rooted tree constructed from n-port switches. This topology comprises $(n/2)^2$ pods, n switches in the Core Layer and $n^2/2$ switches in the Aggregation and Access layers, supporting a total of $n^3/4$ hosts. It exemplifies topologies that can be generated by the simulator using few parameters, without the need to manually specify every individual component and link.

The *Internet*, serves as the medium for all inter-datacenter communications. We model it as an abstract node with infinite bandwidth capacity. We model *Links* as connectors for transmission of network flows. These links can be either simplex or duplex (typically duplex) and are characterized by a finite capacity. This capacity constraint must be considered in routing and Quality of Service (QoS) decisions. We model the *Core Switches* as routing nodes that have internet access, with a fixed number of ports, each operating at a specified maximum speed. The behavior of these switches is governed by routing protocols, energy-saving mechanisms, and Quality of Service (QoS) policies (discussed in detail in Section 3.4). The *Aggregation* and *Access Switches* differ from core switches in that they do not have direct internet access and are typically lower-tier devices, with lower port speeds. In our network model, the hosts handle the computational workload and can also function as routing nodes, as illustrated in Figure 3. We consider the ability to simulate server-centric topologies to be a key advantage of our simulator, distinguishing it from many competitors, as numerous emerging network architectures are based on this principle [50].

We define *Network Flows* by a starting timestamp, an ending timestamp, a source node, a destination node, and an adjustable demand. The routing of these flows, along with the allocated bandwidth, determines the resulting throughput. As demonstrated by the two provided examples, our network model supports both *inter-datacenter* and *intra-datacenter* network flows. Inter-datacenter flows refer to communications between a host within a datacenter and an external machine, whereas intra-datacenter flows, also known as East-West communications, occur between hosts within the same datacenter. The preva-

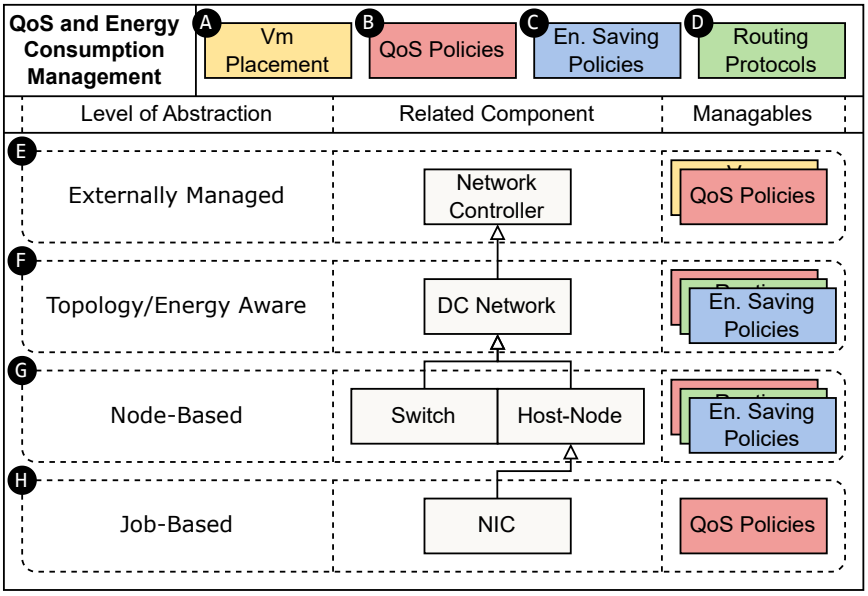


Figure 5: Overview of Policy Management.

lence of these East-West flows is steadily increasing [11], challenging many datacenters and highlighting the importance of evaluating these types of network traffic. Additionally, compared to other datacenter network simulators, our model supports flow distribution across multiple paths, as shown in Figure 4. This enables testing routing protocols such as *Equal Cost Multi-Path* (ECMP), *Dynamic Multi-Path Routing* (DMPR), etc.

3.4 Managing Routing, Energy Saving and QoS Strategies

We describe the various mechanisms through which the simulator enables users to implement custom strategies for routing, energy efficiency, and QoS management. An overview of the policy management model is illustrated in Figure 5. .

The *VM Placement* (A) involves selecting an appropriate host machine, based on resource available, for executing a specific job. The *QoS Policies* (B) define how network resources, typically bandwidth, are allocated to each flow or job, according to some algorithm. We evaluate the impact of different QoS policies in Section 5.2. The *Energy Saving Policies* (C) implement energy-efficient strategies, such as powering down or adjusting maximum speeds of ports and switches, as well as rerouting network flows to reduce overall energy consumption. The *Routing Protocols* (D) determine how flows are routed through the network. These protocols typically balance achieving high performance by distributing the workload across multiple paths, with the need to exclude certain links to enable the energy-saving policies to power down certain devices. As a result, routing protocols and energy-saving strategies are closely correlated, often influencing each other's behavior and effectiveness. We evaluate the impact of different routing protocols in Section 5.3. *Vm Allocation* and *QoS Policies* can be *Externally Managed* (E), and applied through the Network Controller interface. In particular, Vm Allocation can both be pre-defined

by certain trace formats or determined dynamically at runtime through a provisioning process, managed by the compute simulator logic. The Network controller offers real-time insights on the performance of each component of the network, enabling adjustments to data rates based on these metrics. *QoS Policies*, *Routing Protocols* and *Energy Saving Policies* can be managed on the *Topology/Energy Aware* level of abstraction (F). These policies are configurable on the DC Network component and offer access to the entire network state, useful for optimizing performance and energy efficiency across the system. The same three managements can be performed on the *Node Level* (G). In this case, each node makes autonomous decisions based solely on the information it possesses. *QoS Policies* can also be applied on the *Job Level* (H). Each job running on a node possesses full information about the job's connections and network flows, allowing it to manage and optimize its own network resources allocation independentlydently.

3.5 Discrete Network Events

The key events of the simulations are concerning flows. We define three key flow-related events: *flow creation*, *flow termination* and *flow demand update*. Excluding topology changes, failures and hardware speed tuning, we model every network communication and interaction through these three events. All input traces are converted into workloads constructed using these three fundamental events, which trigger the required reevaluations of policies and protocols. Each network event is associated with a timestamp. Adjustments resulting from events occurring at the same timestamp should be performed concurrently. However, advancement of *virtual time* should occur only when the network is considered stable, meaning no further adjustments are needed unless another event occurs. Therefore, monitored metrics can be updated at each virtual time advancement without inconsistencies due to incomplete adjustment. A timeline example of 2 flows and how their events are handled is shown in Figure 6, where the "Compute" corresponds to discrete simulation events which trigger the application of the required policies and protocols based on the updated flow state. Our model does not account for transmission time, as we consider it to have minimal impact on QoS and energy consumption within the datacenter, thereby simplifying the overall model.

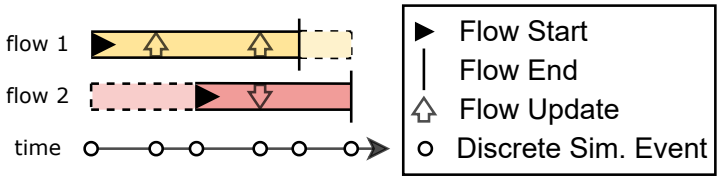


Figure 6: Timeline Example of 2 network flows' events.

4 Prototype Implementation in a state-of-the-art Simulator (RQ2)

This Section outlines the main implementation concepts of our network simulator. The simulator is designed for high parallelism, with each network node operating within its own coroutine. Mechanisms for ensuring consistency during simulation are detailed in Section 4.1, while Section 4.2 provides an overview of the internal execution loop of each node. The simulator's input specification is described in Section 4.3, and the generated output metrics are discussed in Section 4.4. Section 4.5 introduces the OpenDCN REPL environment, which supports interactive experimentation and debugging. An overview of the integration between OpenDCN (network) and OpenDC (compute) is presented in Section 4.6, followed by an example of a combined simulation in Section 4.7.

4.1 Ensuring Network Consistency

Given the high parallelism of our network simulation, which consists in a separate coroutine for each network node, a mechanism is required to guarantee consistency during event driven simulation. We consider the network to be in a *stable* state when all preceding events have been fully processed and their effects have been propagated throughout the network, such that no further computation remains pending.

To manage this, we introduce the concept of a *Stability Barrier*, a structure designed to provide mechanisms for checking, enforcing, or awaiting network stability before or during operations. Each node is equipped with an *Invalidator*, which is linked to the barrier. During simulation, nodes can autonomously mark themselves as stable or unstable based on their processing state. The barrier maintains a collection of invalidators organized in a tree-like hierarchy, where global stability, represented by the stability of the root node, is reached by propagating validated states from the leaves upward. The tree has variable depth, depending on the number of invalidators that are dispensed. To preserve consistency, nodes propagating an update to an adjacent node invalidate the stability state of that adjacent node prior to validating their own. Ensuring global network stability is essential for multiple operations, such as computing a snapshot of the state or advancing the simulation virtual time. The barrier enables parallel processing of network events occurring at the same virtual time instant, while ensuring that all updates are fully propagated before advancing to the next simulation timestamp. Network stability can be either *checked*, wherein the simulation halts if stability is violated during an operation, or *enforced*, wherein the system awaits a stable state and suspends further processing during the operation.

This structure is fundamental to maintaining simulator correctness and is a potential target for future performance optimization efforts.

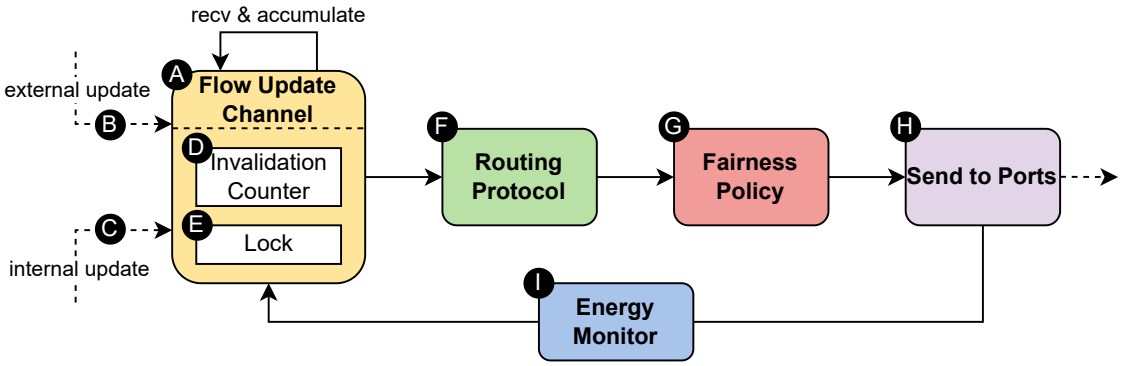


Figure 7: High level Overview of a Node State Machine.

4.2 Node Execution Loop

In our implementation, each node within the network operates within its own coroutine, achieving parallelism. The mechanisms for ensuring consistency and preventing deadlocks in this concurrent environment are detailed in Section 4.1.

A high-level overview of the node's coroutine loop is presented in Figure 7. Each node has its own *Flow Update Channel* (A) which serves as a buffer for incoming flow updates to be processed. When multiple updates are present in the channel, they are aggregated and processed as a batch. This strategy aims to minimize the computational cost of applying routing protocols and fairness policies, and to reduce the likelihood of propagating soon to be obsolete updates. Flow updates may originate from adjacent nodes (B), or internally in case of a node capable of initiating flows C. As explained in Section 4.1, each node is provided with an *Invalidator* to mark itself as unstable while processing updates. A corresponding *Invalidation Counter* (D) is incremented with each incoming update and decremented upon successful processing. A node is considered to be in a stable state if, and only if, the invalidation counter is zero and the coroutine is suspended awaiting new updates. Update processing may be temporarily suspended by acquiring the node's *Lock* (E), for example, during changes to the topology such as node connection/disconnection, or while computing a snapshot of the node state. When no further updates are available in the channel, the node proceeds to process the accumulated batch, starting by applying routing protocols (F) for new flows or in case the routing table was changed. Once the set of ports for propagating flows is determined, fairness policies, such as max-min fairness, are applied (H). The resulting output data rates are then transmitted to adjacent nodes via the corresponding ports (H), ensuring that the stability state of affected neighboring nodes is invalidated prior to proceeding with the next step. Before returning to the initial awaiting state, the *Energy Monitor* (I) is signaled to update its internal statistics, as these may have changed in the loop iteration.

It should be noted that this description presents a simplified overview of the internal node operation. Nonetheless, it serves to clarify the parallel execution model and the mechanism by which events are propagated throughout the network.

```
{
  "type": "fat-tree-specs",
  "coreSwitchSpecs": {
    "numOfPorts": 4,
    "portSpeed": "10Gbps",
    "fairnessPolicy": { "type": "max-min" },
    "portSelectionPolicy": { "type": "ospf" }
  },
  "aggrSwitchSpecs": {
    "numOfPorts": 4,
    "portSpeed": "1Gbps",
    "fairnessPolicy": { "type": "max-min" },
    "portSelectionPolicy": { "type": "ospf" }
  },
  "torSwitchSpecs": {
    "numOfPorts": 4,
    "portSpeed": "500Mbps",
    "fairnessPolicy": { "type": "max-min" },
    "portSelectionPolicy": { "type": "ospf" }
  },
  "hostNodeSpecs": {
    "numOfPorts": 1,
    "portSpeed": "1Gbps",
    "fairnessPolicy": { "type": "max-min" },
    "portSelectionPolicy": { "type": "static-ecmp" }
  }
}
```

(a) Fat Tree (n=4) of Figure 4.

```
{
  "type": "fat-tree-specs",
  "switchSpecs": {
    "numOfPorts": 12,
    "portSpeed": "10Gbps",
    "fairnessPolicy": { "type": "max-min" },
    "portSelectionPolicy": { "type": "ecmp" }
  },
  "hostNodeSpecs": {
    "numOfPorts": 1,
    "portSpeed": "500Kbps",
  }
}
```

(b) Fat Tree (n=12) with 612 nodes (432 hosts).

Figure 8: Example of a Fat Tree topology specification in OpenDC.

4.3 Simulator Inputs

This Section provides a detailed overview of the various input components required by the network simulator. First, Section 4.3.1 describes the methodology for defining a network topology. Next, Section 4.3.2 presents the process of converting diverse, non-standardized trace formats into a format compatible with the simulator. Finally, Section 4.3.3 outlines the configuration of a network simulation scenario, which integrates all input parameters, network configurations, and export settings necessary for conducting simulations.

4.3.1 Building a Network

One of the main requirements of the simulator is to be able to simulate arbitrary network topologies, enabling reuse and sharing of designs. To fulfill these requirements, each network topology is represented in JSON format, which facilitates easy sharing, modification, and reuse. The recursive structure of many topologies allows for a short JSON description, significantly reducing complexity. Additionally, topologies can be imported into the simulator, modified, tested, and subsequently exported as new JSON files, facilitating the process of iterative design and evaluation. In the following, we aim to demonstrate the process of designing DC networks through two simple examples of network topologies along with their respective JSON representations. In Figure 8a you can see the JSON representation of the topology illustrate in Figure 4. This is a clear example of a well known topology that can be built with few parameters. In this example, the topology requires only the specifications of the three distinct types of switches used at different layers of the

Field	Type	Meaning
`timestamp`	`int64`	The milliseconds elapsed from epoch
`transmitter_id*`	`int64`	The id of the transmitter node, if `null` assumed internet
`destination_id*`	`int64`	The id of the destination node, if `null` assumed internet
`net_tx`	`double`	The data-rate of the flow in `Kbps`
`flow_id*`	`int64`	The id of the flow, if `null` only 1 flow is possible between 2 nodes
`duration*`	`int64`	The duration of the flow in `ms`, if `null` the flow will not change until further network events affect it

* = nullable

Table 1: Overview of the network-only Workload Format of the OpenDCN simulator.

fat tree. Alternatively, a single switch configuration can be defined and applied across all layers, as in Figure 8b. The n (which is the main parameter for the recursive algorithm) will be determined based on the smallest number of ports among all switches, in this case 4.

Figure 4.3.1 demonstrates a custom topology that does not leverage the recursive topology builder capabilities of the simulator. The JSON structure first defines all nodes, specifying attributes such as port speed, number of ports, and any default policies or protocols applied in the absence of network-specific policies. Alternatively, network or energy-aware policies can also be defined similarly, but within the network object. Following the node definitions, the links between nodes are outlined, detailing the network connections and their respective link bandwidths.

An alternative method for constructing network topologies through the OpenDCN REPL environment is explored in Section 4.5. The simulator is designed to construct and simulate any conceivable network topology, offering flexibility for creative network designs. This enables users to explore a wide range of configurations and test them within minutes.

4.3.2 Network Workloads

The schema shown in Table 1 has been designed to facilitate building network workloads from any conceivable trace, supporting total control of the flow through updates, as well as through flow durations. It enables inter- and intra-datacenter communications, allowing for two flows between each host (one per direction) as well as multiple flows between two machines when a `flow_id` is specified. This flexibility ensures the generation of precise network workloads from diverse non-standardized input data. As additional constraints, `transmitter_id` and `destination_id` cannot be equal (including `null`, and either the whole `flow_id` column is specified, or it is discarded. Duration does not need to be fully defined for each row, however it is favorable to either convert the non-standardized trace to a workload with the whole duration column define or none.

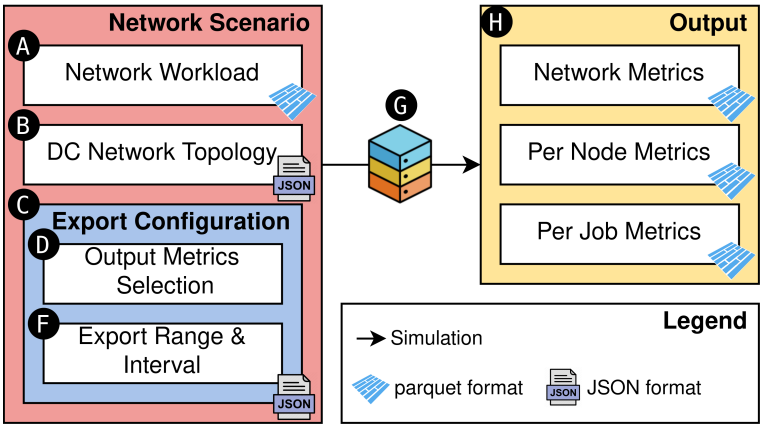


Figure 9: High level view of Network Scenario simulation.

4.3.3 Network Scenario

In Figure 9 we show a high level view of the simulation of a network scenario. The *Network Workload* (A) is a parquet trace, in one of the many formats that the simulator is able to decode, as described in Section 4.3.2. The *Network Topology* (B) is the JSON representation of a DC network, as described in 4.3.1. The *Export Configuration* (D) aggregates all configurable settings concerning the output of the simulation. A subset of all possible *Output Metrics* (D) offered by the simulator can be selected, in order to filter out all the information that may be irrelevant for the user experiment, thus reducing output files sizes.

The user can select to produce output metrics for certain virtual time intervals of the simulation, adjusting the *Export Range*, as well as the export *Granularity* (F). Using these parameters, the simulation (G) can be conducted, ultimately yielding the results (H). These results include a comprehensive set of metrics, including network-level, per-node, and per-job statistics, as detailed in Table 2.

It is important to note that the number of output metrics selected for export can impact the simulator's performance. Therefore, we provide the option to include only the essential metrics in the output, thereby minimizing unnecessary overhead. We use JSON format for input in the simulator because it is a lightweight, human-readable format that offers flexibility in representing complex, nested structures. This makes it ideal for serializing and deserializing components and configurations. We use Parquet format for data export due to its performance advantages, particularly when dealing with large simulation datasets. Parquet's compression reduces storage space and speeds up data transfer, making it well-suited for handling large datasets efficiently.

```
{
    "networkSpecsPath": "net-spec.json",
    "wlPath": "example_net_trace.parquet",
    "exportConfig": {
        "networkExportColumns": [
            "timestamp_absolute",
            "avrg_tput_ratio",
            "power_draw_watt"
        ],
        "nodeExportColumns": [
            "timestamp_absolute",
            "node_throughput_mbps",
            "energy_consumed_joule"
        ],
        "outputFolder": "output/test1",
        "exportInterval": "5min"
    },
    "virtualMapping": true
}
```

(a) Example Network Scenario.

```
10:19:36.265 [INFO] SimNetWorkload -
| === network workload reading schema ===
| message net_wl_reading_schema {
|   optional int64 transmitter_id;
|   required double net_tx;
|   required int64 timestamp (TIMESTAMP(MILLIS,false));
|   optional int64 dest_id;
| }
10:19:36.928 [INFO] NetworkScenario -
| === NETWORK EXPORT CONFIG ===
| Network columns  : timestamp_absolute, ...
| Node columns     : timestamp_absolute, ...
| Export interval  : PT5M
| Output folder    : /home/t0m3x/...
10:19:36.931 [INFO] NetworkScenario -
| === NETWORK WORKLOAD ===
| start instant: 2013-08-12T13:40:46Z
| end instant: 2013-09-11T13:39:58Z
| duration: PT719H59M12S
| num of network events: 1071099
10:19:36.933 [INFO] FatTreeNetwork - building with n=8
10:19:39.815 [INFO] NetworkController -
| === NETWORK INFO ===
| num of core switches: 16
| num of host nodes: 128
| num of nodes: 209 (including INTERNET abstract node)
Simulating network... 100% [=============]
1071099/1071099 (0:00:13 / 0:00:00)
```

(b) Corresponding simulation execution.

Figure 10: Execution of a Network Scenario.

Figure 10 illustrates the complete execution process of a network scenario. The JSON representation of the scenario is shown in Figure 10a, while the corresponding CLI output is presented in Figure 10b. During the setup phase, the simulator provides key information, including the format of the trace data being read, the configuration settings for data export, the start and end times of the simulation, its duration, and a concise overview of the basic network topology. As the simulation progresses, a real-time progress indicator is displayed, providing continuous updates on the remaining execution time. This facilitates user monitoring and provides transparency regarding the runtime behavior of the system. An important aspect to highlight is the flexibility in configuring the data export process. Specifically, if the export column definitions are omitted, the simulator defaults to exporting all available metrics, ensuring a comprehensive capture of system state. In scenarios where the export interval is not explicitly defined, the simulator exports snapshots of both network-wide and individual node states at every unique timestamp encountered in the workload's virtual time. While this configuration guarantees maximum temporal granularity in the exported data, it may also incur a performance overhead due to the increased volume and frequency of export operations.

Name	Unit	Description
network.nodes.count	-	Number of nodes currently part of the network.
network.nodes.hosts[all]	-	Number of hosts currently part of the network
network.nodes.hosts[active]	-	Number of active hosts currently part of the network
network.flows.count	-	Number of active network flows.
network.flows.throughput[tot]	Mbps	The sum of the throughput of all active flows.
network.flows.throughput[%]	%	The total throughput percentage of all active flows.
network.flows.throughput[avg%]	%	The average flow throughput percentage.
node.flows.uptime[average]	ms	The average uptime of active flows in the network.
network.energy.power	W	The current power draw of the network.
network.energy.energy	J	The energy consumed by the network up until now.
node.flows[incoming]	-	Number of flows incoming from adjacent nodes.
node.flows[outgoing]	-	Number of flows outgoing to adjacent nodes.
node.flows[generating]	-	Number of flows being generated by this node.
node.flows[consuming]	-	Number of flows being consumed by this node.
node.flows.throughput[tot]	Mbps	The total throughput on the node.
node.flows.throughput[%]	%	Total throughput percentage in the node.
node.flows.throughput[avg%]	%	Average throughput percentage of flows traversing the node.
node.flows.throughput[min%]	%	Min throughput percentage among flows traversing the node.
node.flows.throughput[max%]	%	Max throughput percentage among flows traversing the node.
node.flows.uptime[avg]	ms	The average uptime of active flows traversing the node.
nodes.flows.energy.power	W	The current power draw of the node.
nodes.flows.energy.energy	J	The energy consumed by the node up until now.
job.flows[generating]	-	Number of flows being generated by this job.
job.flows[consuming]	-	Number of flows being consumed by this job.
job.flows.throughput[tot]	Mbps	The total throughput on the job network interface.
job.flows.throughput[%]	%	The total throughput percentage on the job network interface.
job.flows.throughput[min%]	%	Min throughput percentage among flows generated by the job.
job.flows.throughput[max%]	%	Max throughput percentage among flows generated by the job.
job.flows.uptime[average]	ms	The average uptime of active flows generated by the job.

Table 2: Metrics exposed by the OpenDCN Simulator.

4.4 Output Metrics

We expose telemetry data amounting to 30+ different metrics. The most relevant are highlighted in Table 2, we list the representative [19, 43] metrics relevant for this work. Figure 9 shows a high level overview of the simulation process, focusing on inputs and outputs.

4.5 Interactive Network Simulation

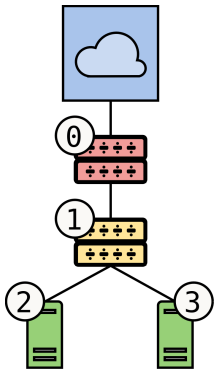
The REPL environment provides an interactive environment for event-driven simulations as well as for building and modifying components and topologies. It allows users to rapidly test newly defined policies and protocols in a more debugging-friendly environment. This feature is particularly valuable for educational purposes and facilitates plug-and-play simu-

Objective	Stakeholders
Network Design & Optimization	DC Operators, Researchers
Testing of new Policies & Protocols	Researchers, Academics
Interactive Simulation & Immediate Feedback	Students, Trainees
Edge Case Analysis	DC Operators, Security Analysts

Table 3: REPL environment usecases.

```
> core switch 0 1Gbps 2 ECMP MaxMin
11:28:52.223 [INFO] NEW_SWITCH - successfully added
> switch 1 100Mbps 3 ECMP MaxMin
11:29:03.758 [INFO] NEW_SWITCH - successfully added
> host 2 1Gbps 1 ECMP MaxMin
11:29:08.967 [INFO] NEW_HOST - successfully added
> host 3 1Gbps 1 ECMP MaxMin
11:29:08.967 [INFO] NEW_HOST - successfully added
> link 0 - 1 100Mbps
11:29:23.395 [INFO] NEW_LINK - link successfully created
> link 1 - 2 100Mbps
11:29:23.395 [INFO] NEW_LINK - link successfully created
> flow 2 -> 0 1Gbps
11:50:57.380 [INFO] NEW_FLOW - successfully created
> flows
11:50:59.466 ] [INFO] SHOW_FLOWS -
| ==== Flows ====
| id   sender    dest      demand              throughput
| 0    2         0         1.000 Gbps          100.00 Mbps
> flow 3 -> 0 1Gbps
11:58:15.658  [INFO] NEW_FLOW - successfully created
11:58:16.846 [INFO] SHOW_FLOWS -
| ==== Flows ====
| id   sender    dest      demand              throughput
| 0    2         0         1.000 Gbps          50.000 Mbps
| 1    3         0         1.000 Gbps          50.000 Mbps
> export myfolder/myfile.json
```

(a) Command Line



(b) Corresponding Topology

Figure 11: Example Usecase of the REPL Environment.

lations, exploring network behaviors with ease. The sandbox environment allows to trigger any event that the normal simulation can make happen and more.

Figure 11a shows the use of the sandbox environment through CLI. For this example, the environment is initially empty, although it can be configured to load any previously defined components into the sandbox. A simple network topology comprising two switches and two hosts with specific characteristics is constructed, and is visually displayed in Figure 11b. To validate the setup, two data flows are initiated, and basic network information is displayed. The network is then saved to a JSON file for further use. Keep in mind that complete metrics of the state of the network and its nodes can be exported at any given time during sandbox experimentation. This simplified example demonstrate one of many uses of the environment, which allows the user to have total control over the events of the simulation and its virtual time, which can be advanced through a dedicated command.

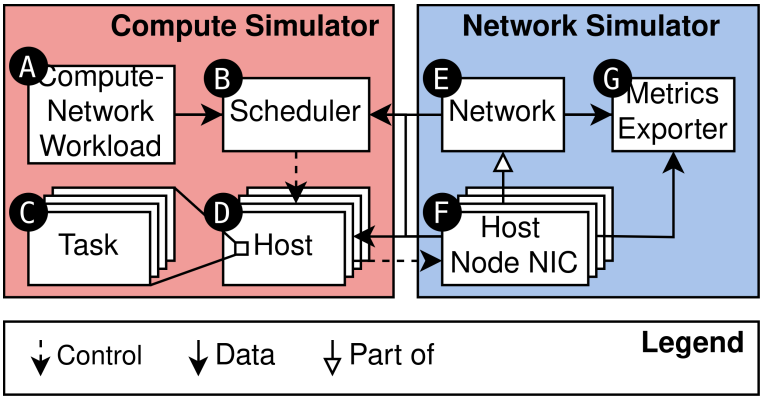


Figure 12: Network-Compute Integration Overview.

4.6 Integration with OpenDC

We extend OpenDC [28, 36, 37], an open-source datacenter simulation platform, which incorporates advanced models for emerging cloud-datacenter technologies and applications, including serverless computing with FaaS deployment and TensorFlow-based machine learning. By integrating network simulation on top of OpenDC, instead of creating a stand-alone tool, we ensure the long term support of the OpenDC platform. Additionally, the resulting tool can offer a closer representation of real life scenarios with combined simulation of compute and networking and their interactions.

```
{
    "networkSpecsPath": "test_network.json",
    "clusters":
  [{
      "name": "C01",
      "hosts" : [{
          "count": 128,
          "cpu": {
            "coreCount": 8,
            "coreSpeed": "3.2 Ghz"
          },
          "memory": {
            "memorySize": "128e3 MiB",
            "memorySpeed": "1 Mhz"
          },
          "powerModel": {
            "modelType": "linear",
            "power": "400 Watts",
            "maxPower": "1 KW",
            "idlePower": "0.4W"
          }
      }]
  }]
}
```

(a) Define 128 hosts to be mapped to `test_network`.

```
{
  "type": "fat-tree-specs",
  "switchSpecs": {
    "numOfPorts": 8,
    "portSpeed": "10Gbps"
  }
}
```

(b) Define `test_network` (128 host nodes).

Figure 13: Setting Up Topology for Compute-Network Simulation.

Section	Focus	Independent Variables
	Feature Evaluation	
§5.2	QoS Policies	QoS policy used (MaxMin, FCFS, ...)
§5.3	Routing Protocol	Routing protocol used (ECMP, OSPF, ...)
§5.4	Topology	Topology used (Fat-Tree, FlatNet, Spine-Leaf...)
	Performance Evaluation	
§5.5	Scalability	Workload Size, Number of Nodes, ...

Table 4: Experiments Overview.

4.7 Combined Network and Compute Simulation with OpenDC

In Figure 12 we present an overview of the integration of the network simulator with the OpenDC compute simulator. Keep in mind this is an extremely simplified view of the system. In order to execute a combined compute-networking simulation the input *Workload* (Ⓐ) must be in one of the many formats compatible with the OpenDC platform and include networking information. The *Scheduler* (Ⓑ) is responsible for determining the allocation of virtual machines (VMs), making decisions based on predefined policies and weighted resource availability. Each *Task* (Ⓒ) represents a computational job, service, or application process that is executed within a virtual environment hosted on a physical machine. The physical machines, referred to as *Hosts* (Ⓓ), are capable of running multiple VMs concurrently. Each host is mapped to a node in the *Network* (Ⓔ), and to its *Network Interface* (Ⓕ). Each job can interact with the network through the interface, and receives feedback on its network flows. As detailed in Figure 5, this feedback enables the application of Quality of Service (QoS) policies at both the task and host levels. Consistent with the approach used in network-only simulations, the *Metric Exporter* (Ⓖ) records the state of the network and its nodes, in accordance with the specified simulation configuration, and exports the resulting data in Parquet format

5 Evaluation using Large-Scale Simulation (RQ3)

To demonstrate the effectiveness of our simulator, we conduct a set of experiments aimed at validating its functionality and assessing the influence of individual components and policies on overall datacenter energy consumption and QoS. Furthermore, we analyze the simulator's worst-case performance and scalability by varying independent variables, thereby offering deeper insights into its operational limits and highlighting potential areas for future optimization.

5.1 Main Findings

Our experimental findings indicate that the simulator effectively captures the impact of various architectural and policy-level components on datacenter behavior. Specifically, we confirm that adopting a FatTree topology yields performance improvements over a traditional tree-based topology. Moreover, the use of Max-Min fairness for resource allocation

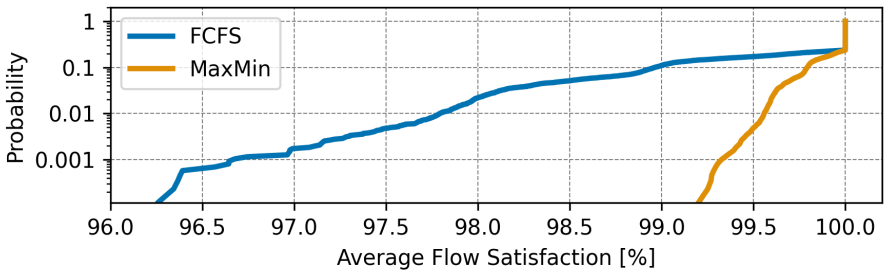


Figure 14: MaxMin vs. FCFS Tail Distribution of Average Flow Satisfaction.

results in higher levels of user demand satisfaction when compared to a simple first-come, first-served (FCFS) approach. In addition, while Equal-Cost Multi-Path (ECMP) routing incurs greater energy consumption than Open Shortest Path First (OSPF), it achieves a superior network power effectiveness (NPE) ratio, suggesting its greater suitability for the simulated scenario. Finally, our evaluation reveals that the simulator delivers acceptable performance under a range of conditions, although scalability and runtime efficiency remain potential areas for future enhancement

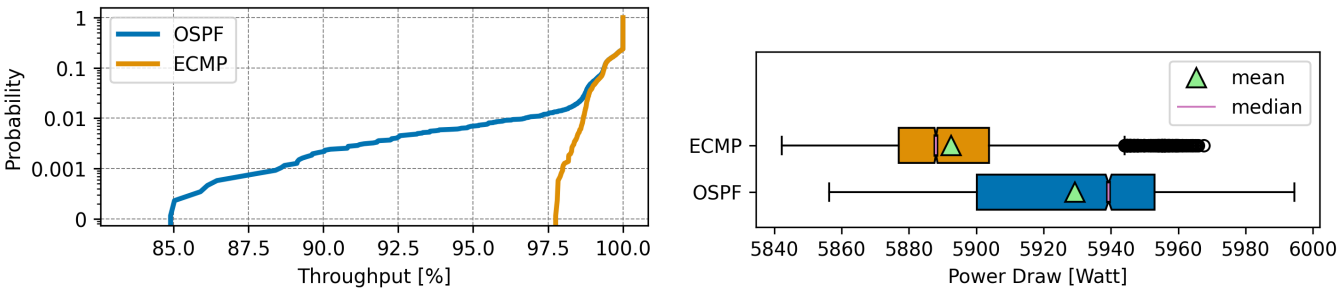
5.2 Evaluating Fairness Policies

Experiment Setup: This experiment aims to compare fairness of different policies (see Section 2.5), in particular we focus on First Come First Served and Max Min Fairness [38], but the experiment can be extended with any fairness policy.

Results: Figure 14 shows the tail distribution of the throughput percentage (throughput/demand) for the two different setups. As expected MaxMin achieve better average flow satisfaction, with FCFS yielding a lower average flow throughput percentage ($\approx -7.1\%$). This reduction can be attributed to the fact that high-demand flows can prevent lower-demand ones from reaching their required demand, lowering the average.

5.3 Evaluating Routing Protocols

Experiment Setup: This experiments aims to compare throughput and energy consumption metrics of different routing protocols that are widely used in practice (see Section 2.5), with otherwise identical setups. In particular this experiment focuses on ECMP [29] and OSPF [53]. The evaluation employs the Bitbrains workload on a FatTree network topology configured with 128 hosts and 1 Gbps switches. The workload is derived from the fastStorage dataset of the GWA-T-12 Bitbrains workload trace [47]. Bitbrains, a managed hosting and enterprise computation service provider, supplies this dataset, which includes traces from 1,250 virtual machines (VMs) deployed across software applications hosted within Bitbrains' datacenter infrastructure.



(a) Throughput tail distribution.

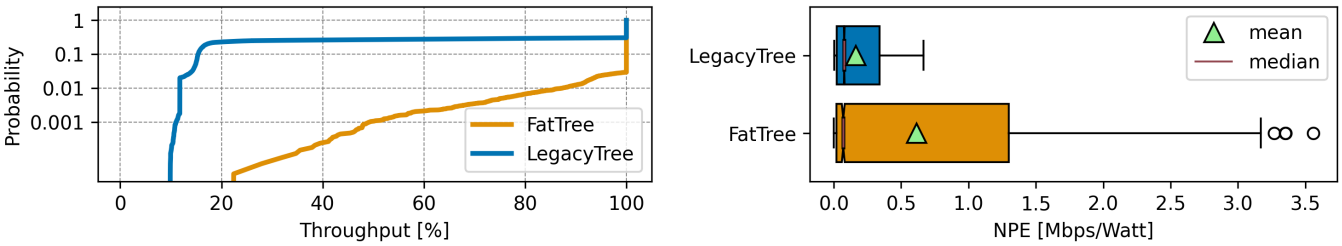
(b) Power draw at uniform intervals.

Figure 15: OSPF and ECMP Comparison

Results: Figure 15a shows that in our simulation ECMP achieves higher throughput, with a reduction in peak packet drop rates by approximately 65% compared to OSPF. Additionally, this increased throughput and overall link utilization does not adversely affect network energy consumption. As shown in Figure 15b, we observe a slight decrease in energy consumption ($\approx .5\%$). Regarding NPE (Section 2.7), the values observed for ECMP were nearly identical to those of OSPF, $2.32 Mb/W$ and $2.30 Mb/W$ respectively, with a marginal reduction of $\approx 0.7\%$. This aligns with expectations, given that congestion spikes, which influence NPE and CNEE metrics, are relatively infrequent over extended observation periods.

5.4 Evaluating Topologies

Experiment Setup: This study aims to evaluate the impact of different network topologies (see Section 2.4) on QoS and energy consumption in datacenters, while also demonstrating the effectiveness of our simulator in assessing these metrics. Specifically, we focus on comparing the advancements offered by the FatTree topology [7], relative to a legacy tree-based datacenter architecture. For this experiment, both the FatTree and legacy tree topologies are simulated in configurations with 250 host nodes and 1 Gbps switches on all layers. We employ Max-Min Fairness for resource allocation and Equal-Cost Multi-Path (ECMP) routing to ensure balanced load distribution across network paths.



(a) Throughput tail distribution.

(b) NPE at uniform intervals.

Figure 16: LegacyTree vs. FatTree.

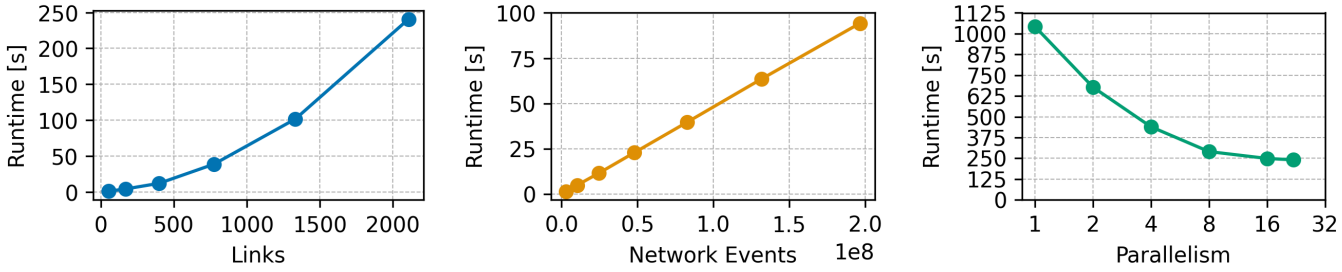
Results: Figure 16 shows some of the results of the experiment. FatTree topology exhibits a 61% higher power consumption relative to the Tree topology, this is due to higher number of network components, in particular switches, required for such topology. Despite this increased energy draw, FatTree achieves higher network performance, with an average throughput improvement of 73.1% over the Tree topology. Additionally, Figure 16a highlights a notable disparity in traffic demand fulfillment, where FatTree consistently maintains a high fullfilment percentage. These results are likely due to the Equal-Cost Multi-Path (ECMP) routing in the FatTree architecture, which optimizes link utilization and reduces bottlenecks. In contrast, the conventional hierarchical structure of the Tree topology tends to lead to link saturation.

Figure 16b indicates that the increased power consumption from FatTree's larger number of network devices is compensated by its capacity to accommodate higher traffic demand, resulting in an overall gain in NPE (defined in Eq 6). It is important to note that these results may vary with different workloads. For workloads with low bandwidth demands, Tree topology links are less likely to become saturated, reducing or even eliminating the performance advantages typically seen with FatTree topologies.

Regarding NPUE (defined in Eq 5), and calculated based on both host and network energy consumption as total IT energy use, FatTree achieved a NPUE of approximately 4.68, while the Tree topology reached around 6.92, indicating a 48% higher effectiveness.

5.5 Scalability Analysis

Experiment Setup: We evaluate the scalability of our simulator with respect to three key factors: the number of links in the network, the workload size, and the level of available parallelism. To isolate the impact of each variable, all other parameters are held constant while we vary one independent variable at a time. For each configuration, we measure the average simulation runtime over 20 runs. All experiments are conducted on a FatTree topology using ECMP routing. It is important to note that, in a FatTree topology, increasing the number of links inherently increases the number of nodes. However, we consider the number of links to be a more direct indicator of the network's complexity. All runtime are measured while exporting all available metrics, ensuring worst case scenario. For the workload size and parallelism experiments, we employ a FatTree topology comprising over 600 hosts. All runtimes are measured while exporting the full set of available simulation metrics, as an upper bound on the simulation runtime for the given inputs.



(a) Runtime vs. Link Count. (b) Runtime vs. Event Count. (c) Runtime vs. Parallelism.

Figure 17: Scalability of OpenDCN Simulator.

Results: As shown in Figure 17, the simulator exhibits a gradually increasing exponential runtime relative to the number of links in the network. In contrast, as expected, it demonstrates linear scaling with respect to workload size. While larger topologies may benefit more significantly from increased parallelism, the performance gains in this particular setup appear to plateau at around 16 threads, as illustrated in Figure 17c.

6 Conclusions & Future Work

In Section 6.1, we highlight the main contributions of this paper and the insights gained from the experiments. Section 6.2 discusses potential areas for future improvements and enhancements to the simulator.

6.1 Conclusions

In Section 3, we address **RQ1**, presenting our design approach for a discrete network simulator capable of simulating a wide range of topologies. It supports the integration of various routing protocols and fairness policies, which can be easily added, while maintaining flexibility for integration with a compute simulator for compute-network simulations. In Section 4, we address **RQ2**, providing a high-level overview of the key concepts behind the network simulator implementation. This includes details on the inputs, outputs, and how to leverage the interactive REPL environment. We also provide an overview of how to conduct a combined compute-network simulation by integrating the network simulator with OpenDC, a state-of-the-art datacenter simulator [28, 36], along with an example of such a simulation. Finally, in Section 5, we address **RQ3**, demonstrating the usefulness of our simulator both in isolation and in combination with a datacenter compute simulator. We use the simulator to examine the impact of various routing protocols, fairness policies, and topologies, both in network simulations and compute-network simulations, on energy consumption and QoS. Additionally, we perform a scalability analysis, highlighting both positive and less favorable results, which encourage future improvements in terms of performance and scalability.

6.2 Future Work

We envision 5 main areas of future work to further improve the simulator:

1 **Performance Enhancements:** We plan to implement various performance optimizations. There is significant potential for improvement, and our goal is to develop a simulator that not only adheres to modern engineering standards and maintains high flexibility but also offers competitive performance compared to other state-of-the-art simulators.

2 **GUI REPL:** Although the REPL environment is already user-friendly, we aim to further enhance it by integrating a Graphical User Interface (GUI). This addition will promote adoption in educational contexts by providing visually interactive datacenter simulations, where the consequences of user actions are immediately reflected on the screen. The GUI will also serve as a powerful tool for testing and debugging components, thereby improving the overall user experience.

3 **Fine Grained Compute-Network Resource Dependencies:** We intend to deepen the integration between the network simulator and OpenDC, aiming for a more seamless and efficient simulation environment. This will allow for better-defined resource dependencies between the two modules, resulting in highly precise results and improved performance in compute-network simulations, an important, though often overlooked, area that has the potential to yield valuable insights.

4 **Expansion of Pre-built Components and Policies:** We aim to expand the library of pre-built, available components and built-in, recursively defined network topologies. Additionally, we plan to increase the number of policies provided out-of-the-box, along with detailed experimental evaluations of their effectiveness in different simulation scenarios.

5 **Educational Materials and Documentation:** We plan to develop comprehensive educational materials to facilitate adoption of the simulator. This will include detailed documentation aimed at both users and developers. For users, we will offer tutorials and interactive workshops, while for developers, we will provide extensive documentation on the simulator's architecture, covering both high-level concepts and low-level components, enabling further extension and contribution to the open-source project.

References

[1] ACM artifact review and badging. `https://www.acm.org/publications/policies/artifact-review-and-badging-current`.

[2] Nokia the global network traffic report. `https://onestore.nokia.com/asset/21366`.

[3] Statista volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025. `https://www.statista.com/statistics/871513/worldwide-data-created`.

[4] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu. Energy proportional datacenter networks. *SIGARCH Comput. Archit. News*, 38(3):338–347, jun 2010.

[5] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly. Symbiotic routing in future data centers. *SIGCOMM Comput. Commun. Rev.*, 40(4):51–62, aug 2010.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Hedera: Dynamic Flow Scheduling for Data Center Networks.*, pages 281–296, 07 2010.

[7] M. A. Al-Fares. *A scalable, adaptive, and extensible data center network architecture.* University of California, San Diego, 2012.

[8] M. Alizadeh and T. Edsall. On the data path performance of leaf-spine datacenter fabrics. In *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, pages 71–74, 2013.

[9] M. Antic, N. Maksic, P. Knezevic, and A. Smiljanic. Two phase load balanced routing using ospf. *IEEE Journal on selected areas in Communications*, 28(1):51–59, 2009.

[10] M. Antic and A. Smiljanic. Routing with load balancing: increasing the guaranteed node traffics. *IEEE communications letters*, 13(6):450–452, 2009.

[11] F. Benamrane, M. Ben mamoun, and R. Benaini. An east-west interface for distributed sdn control plane: Implementation and evaluation. *Computers Electrical Engineering*, 57:162–175, 2017.

[12] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica. Surviving failures in bandwidth-constrained datacenters. *SIGCOMM Comput. Commun. Rev.*, 42(4):431–442, aug 2012.

[13] T. Daim, J. Justice, M. Krampits, M. Letts, G. Subramanian, and M. Thirumalai. Data center metrics: An energy efficiency model for information technology managers. *Management of Environmental Quality: An International Journal*, 20(6):712–731, 2009.

[14] M. Dayarathna, Y. Wen, and R. Fan. Data center energy consumption modeling: A survey. *IEEE Communications Surveys Tutorials*, 18(1):732–794, 2016.

[15] P. Dechamps. The iea world energy outlook 2022 – a brief analysis and implications. *The European Energy and Climate Journal*, 11(3):100 – 103, 2023.

[16] A. Di Domenico, G. Perna, M. Trevisan, L. Vassio, and D. Giordano. A network analysis on cloud gaming: Stadia, geforce now and psnow. *Network*, 1(3):247–260, 2021.

[17] N. Feamster, J. Rexford, and E. Zegura. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[18] A. Feldmann, O. Gasser, F. Lichtblau, E. Pujol, I. Poese, C. Dietzel, D. Wagner, M. Wichtlhuber, J. Tapiador, N. Vallina-Rodriguez, O. Hohlfeld, and G. Smaragdakis. The lockdown effect: Implications of the covid-19 pandemic on internet traffic. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 1–18, New York, NY, USA, 2020. Association for Computing Machinery.

[19] C. Fiandrino, D. Kliazovich, P. Bouvry, and A. Y. Zomaya. Performance metrics for data center communication systems. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 98–105, 2015.

[20] C. Fiandrino, D. Kliazovich, P. Bouvry, and A. Y. Zomaya. Performance and energy efficiency metrics for communication systems of cloud computing data centers. *IEEE Transactions on Cloud Computing*, 5(4):738–750, 2017.

[21] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2011 Conference*, pages 350–361, 2011.

[22] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. Vl2: A scalable and flexible data center network. *ACM SIGCOMM Computer Communication Review*, 39:51–62, 01 2011.

[23] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, page 63–74, New York, NY, USA, 2009. Association for Computing Machinery.

[24] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, page 75–86, New York, NY, USA, 2008. Association for Computing Machinery.

[25] A. Hammadi and L. Mhamdi. A survey on architectures and energy efficiency in data center networks. *Computer Communications*, 40:1–21, 2014.

[26] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: saving energy in data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, page 17, USA, 2010. USENIX Association.

[27] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen. Gaminganywhere: an open cloud gaming system. In *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, page 36–47, New York, NY, USA, 2013. Association for Computing Machinery.

[28] A. Iosup, G. Andreadis, V. Van Beek, M. Bijman, E. Van Eyk, M. Neacsu, L. Overweel, S. Talluri, L. Versluis, and M. Visser. The opendc vision: Towards collaborative datacenter simulation and exploration for everybody. In *2017 16th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 85–94. IEEE, 2017.

[29] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene. Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, page 149–160, New York, NY, USA, 2014. Association for Computing Machinery.

[30] K. Kant. Data center evolution: A tutorial on state of the art, issues, and challenges. *Computer Networks*, 53(17):2939–2965, 2009.

[31] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu. Ficonn: Using backup port for server interconnection in data centers. In *IEEE INFOCOM 2009*, pages 2276–2285, 2009.

[32] D. Lin, Y. Liu, M. Hamdi, and J. Muppala. Flatnet: Towards a flatter data center network. In *2012 IEEE Global Communications Conference (GLOBECOM)*, pages 2499–2504, 2012.

[33] Y. Liu, X. Wei, J. Xiao, Z. Liu, Y. Xu, and Y. Tian. Energy consumption and emission mitigation prediction based on data center traffic and pue for global data centers. *Global Energy Interconnection*, 3(3):272–282, 2020.

[34] N. Maksic and A. Smiljanic. Improving utilization of data center networks. *IEEE Communications Magazine*, 51(11):32–38, 2013.

[35] C. Malone and C. Belady. Metrics to characterize data center & it equipment energy use. In *Proceedings of the Digital Power Forum, Richardson, TX*, volume 35, 2006.

[36] F. Mastenbroek, G. Andreadis, S. Jounaid, W. Lai, J. Burley, J. Bosch, E. V. Eyk, L. Versluis, V. van Beek, and A. Iosup. Opendc 2.0: Convenient modeling and simulation of emerging technologies in cloud datacenters. *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 455–464, 2021.

[37] F. Mastenbroek, T. D. Matteis, V. van Beek, and A. Iosup. Radice: A risk analysis framework for datacenters. *IEEE Transactions on Cloud Computing*, 2023.

[38] D. Nace and M. Pioro. Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial. *IEEE Communications Surveys Tutorials*, 10(4):5–17, 2008.

[39] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39(4):39–50, Aug. 2009.

[40] M. Noormohammadpour and C. S. Raghavendra. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Communications Surveys and Tutorials*, 20(2):1492–1525, 2018.

[41] O. Popoola and B. Pranggono. On energy consumption of switch-centric data center networks. *J. Supercomput.*, 74(1):334–369, jan 2018.

[42] A. Predescu and M. Mocanu. A data driven survey of video games. In *2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–6, 2020.

[43] V. D. Reddy, B. Setz, G. S. V. R. K. Rao, G. R. Gangadharan, and M. Aiello. Metrics for sustainable data centers. *IEEE Transactions on Sustainable Computing*, 2(3):290–303, 2017.

[44] D. Reinsel, J. Gantz, and J. Rydning. Data age 2025: The evolution of data to life-critical. don't focus on big data. *2*, 2017.

[45] Y. Shang, D. Li, and M. Xu. Energy-aware routing in data center network. In *Proceedings of the First ACM SIGCOMM Workshop on Green Networking*, Green Networking '10, page 1–8, New York, NY, USA, 2010. Association for Computing Machinery.

[46] Y. Shang, D. Li, J. Zhu, and M. Xu. On the network power effectiveness of data center architectures. *IEEE Transactions on Computers*, 64(11):3237–3248, 2015.

[47] S. Shen, V. Van Beek, and A. Iosup. Statistical characterization of business-critical workloads hosted in cloud datacenters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 465–474, 2015.

[48] M. Vesović, A. Smiljanić, and D. Kostić. Fast and scalable routing protocols for data center networks. *Digital Communications and Networks*, 9(6):1340–1350, 2023.

[49] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford. Central control over distributed routing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 43–56, 2015.

[50] G. Wang, Y. Zhang, J. Yu, M. Ma, C. Hu, J. Fan, and L. Zhang. Hs-dcell: A highly scalable dcell-based server-centric topology for data center networks. *IEEE/ACM Transactions on Networking*, 32(5):3808–3823, 2024.

[51] T. Wang, Z. Su, Y. Xia, B. Qin, and M. Hamdi. Novacube: A low latency torus-based network architecture for data centers. *2014 IEEE Global Communications Conference, GLOBECOM 2014*, pages 2252–2257, 02 2015.

[52] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao. Carpo: Correlation-aware power optimization in data center networks. In *2012 Proceedings IEEE INFOCOM*, pages 1125–1133, 2012.

[53] W. Wollman and Y. Barsoum. Overview of open shortest path first, version 2 (ospf v2) routing in the tactical environment. In *Proceedings of MILCOM '95*, volume 3, pages 925–930 vol.3, 1995.