國立清華大學電機資訊學院資訊工程研究所
碩士論文
Department of Computer Science
College of Electrical Engineering and Computer Science
National Tsing Hua University
Master Thesis

在建構於物聯網之上的智慧環境中最佳化網路數位孿生控制器
Optimizing Network Digital-Twin Controllers for Internet-of-Things Instrumented Smart Environments

吳仕群
Chih-Chun Wu

學號：111062668
Student ID:111062668

指導教授：徐正炘 博士
Advisor: Cheng-Hsin Hsu, Ph.D.

中華民國 113 年 8 月
August, 2024

國立清華大學
資訊工程研究所

碩士論文

在建構於物聯網之上的智慧環境中最佳化網路數位學生控制器

吳仕群

112

Abstract

The rapid deployment of Internet-of-Things (IoT) devices in smart environments such as smart campuses and cities necessitates robust Quality-of-Service (QoS) management across heterogeneous networks. In this thesis, we extend the concept of Network Digital Twin (NDT) to networked IoT devices, presenting a Network Digital Twin Controller (NDTC) that enhances the functionality and performance of smart environments. Our NDTC addresses key challenges by creating Digital Twins (DTs) of Physical Twins (PTs), synchronizing their states, and performing QoS-related what-if analysis. Specifically, we build a DT-enabled IoT-instrumented smart environment by utilizing an open-source Software-Defined Network (SDN) controller. We formulate and solve the state synchronization problem using our proposed Optimal Update (OU) and Gradient-driven Update (GU) algorithms, carefully adjusting the update frequency and data granularity to minimize DT/PT state deviation within given network bandwidth budgets. We also formulate and address the what-if analysis problem by selecting optimal what-if analyzers using our Optimal Selection (OS) algorithm for the most accurate QoS predictions under a given computing time budget. Our extensive experiments on a real testbed demonstrate the merits of our proposed solution: (i) our developed NDTC and algorithms meet the functional requirements, (ii) our OU and GU algorithms significantly reduce the state deviation between PTs and DTs, (iii) our OS algorithm largely reduces the prediction errors of what-if analysis, and (iv) all our proposed algorithms incur acceptable overhead.

中文摘要

隨著物聯網（IoT）設備在智慧環境中的快速部署，如智慧校園和城市，對於異質網路的服務品質（QoS）管理需求日益增長。在本論文中，我們將網路數位孿生（NDT）的概念擴展到網路化的物聯網設備，提出了一種增强智慧環境功能和性能的網路數位孿生控制器（NDTC）。我們的NDTC通過創建實體孿生（PT）的數位孿生（DT）、同步它們的狀態以及進行與QoS相關的假設分析，解決了關鍵挑戰。具體而言，我們利用開源軟體定義網路（SDN）控制器構建了一個DT支持的物聯網儀器化智慧環境。我們使用我們提出的最佳更新（OU）和梯度驅動更新（GU）演算法來解決狀態同步問題，通過調整更新頻率和資料粒度，在給定的網路頻寬預算内最小化DT/PT狀態偏差。我們還通過使用最佳選擇（OS）演算法選擇最優的假設分析器來解決假設分析問題，以在給定的計算時間預算内進行最準確的QoS預測。我們在真實測試平台上進行的實驗展示了我們所提出解決方案的優點：（i）我們開發的NDTC和演算法滿足功能需求，（ii）我們的OU和GU演算法顯著減少了PT與DT之間的狀態偏差，（iii）我們的OS演算法大幅減少了假設分析的預測誤差，（iv）所有我們提出的演算法均帶來可接受的資源消耗。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Increasingly more Internet-of-Things (IoTs) devices have been deployed in smart environments, such as smart rooms, buildings, campuses, communities, and cities, to collect data in surrounding areas and perform actions dictated by innovative applications, like illegal parking detection, adaptive streetlight control, and air pollution mitigation. Such smart environments utilize heterogeneous wireless networks, including WiFi, 5G cellular, LoRa, and Bluetooth, for data exchange. Networked devices, e.g., hosts, routers, switches, gateways, and middleboxes, are deployed in IoT-instrumented smart environments for interconnecting IoT devices to edge/cloud servers through heterogeneous wireless and wired networks.

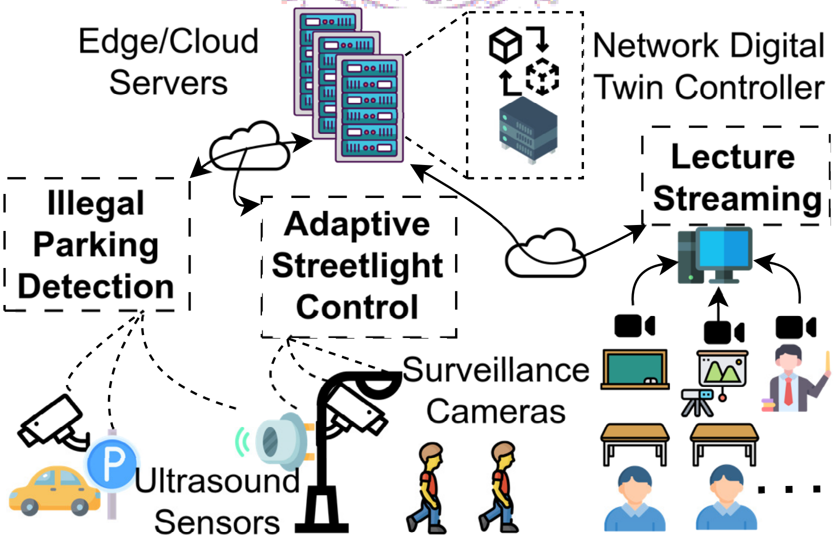

Edge/Cloud Servers
Network Digital Twin Controller
Lecture Streaming
Illegal Parking Detection
Adaptive Streetlight Control
Surveillance Cameras
Ultrasound Sensors
. . .

Figure 1.1: A sample DT-enabled smart campus.

Fig. 1.1 shows a sample smart environment–a smart campus, which hosts three applications: illegal parking detection, adaptive streetlight control, and lecture streaming. Illegal parking detection uses surveillance cameras and ultrasound sensors to recognize illegally parked cars. Adaptive streetlight control also utilizes the same sensors to count the number of pedestrians for suitable streetlight brightness. Lecture streaming multicasts multiple video feeds from the blackboard, slideshow, and audience cameras in a classroom for online attendees. Administrators of smart environments, such as university administration, need to specify the Quality-of-Service (QoS) requirements to ensure the functionality and performance of these applications [18]. QoS requirements can be specified in various metrics, such as bandwidth, delay, and packet loss rate at each networked device.

Guaranteeing these QoS requirements of each application is no easy task as a large number of physical devices, such as hosts, links, sensors, and actuators, are involved, and the smart environments are highly dynamic. A naive way to concurrently support multiple applications is by asking each of them to interface with all the devices. Doing so, however, leads to duplicated deployment and operational investments in hardware and software. A better solution is to construct a unified platform that collects the latest states from the physical devices and running applications. By doing so, individual applications no longer need to interface with physical devices, avoiding undesirable application silos [57].

Another challenge for administrators is to determine how to upgrade a smart environment when the QoS requirements cannot be met. Take lecture streaming as an example, when the backhaul of the classroom network has insufficient bandwidth, the university administration needs to propose a cost-effective upgrade plan, leading to multiple what-if queries, e.g., "would adding an extra optical fiber from the classroom to a nearby router resolve the QoS issues?" A straightforward solution is trial-and-error, which is expensive, time-consuming, and error-prone. A better solution is to integrate the emerging concept of Digital Twin (DT) [11, 17, 20, 28, 29, 47, 50, 53, 54, 56] into smart environments to answer the what-if queries from administrators. More specifically, DT mirrors each real-world physical device, referred to as a Physical Twin (PT) with all its states into a digital representation referred to as a DT [17, 20, 29]. Incorporating DT with IoT-instrumented smart environments has several benefits, including: (i) live device states shared by multiple applications, (ii) what-if analysis based on live data feeds, and (iii) Artificial Intelligence (AI) driven predictions based on historical device states, among others.

In this thesis, we strive to build DT-enabled IoT instrumented smart environments, which can be seen as extending the idea of Network Digital Twin (NDT) [3, 16, 47, 50, 56] to networked IoT devices. Our core task is developing an NDT Controller (NDTC), whose jobs include: (i) creating DTs as the foundation, (ii) synchronizing DT/PT states for multiple applications, and (iii) performing what-if analysis for administrators. We develop an NDTC and optimize it by addressing two key problems: (i) the state synchronization problem to find the best balance between the update frequency and state granularity of individual PTs so as to minimize the DT/PT state deviation under given network bandwidth budgets; and (ii) the what-if analysis problem to select the best what-if analyzer for each query from the heterogeneous ones built upon the Machine Learning (ML) algorithm, queuing theory, network simulator, and network emulator given a computing time budget.

Our extensive experiments demonstrate that: (i) our developed NDTC and optimization algorithms meet the functional requirements, (ii) our proposed optimal and efficient state synchronization algorithms significantly reduce the state deviation in a real testbed by up to 99.49% and 98.92%, (iii) our proposed optimal what-if analysis algorithm significantly reduces the prediction error of what-if queries by more than 83.63%, and (iv) our proposed algorithms terminate fast without incurring high overhead.

Another challenge of NDTCs is the capability to perform what-if analysis, which allows administrators to predict the impact of various network configurations and operational scenarios on the Quality of Service (QoS) metrics. We formulate the what-if analysis problem and develop an Optimal Selection (OS) algorithm. This algorithm intelligently selects a what-if analyzer to provide the most accurate QoS predictions within a given computing time budget. This capability is crucial for making informed decisions regarding network upgrades and reconfigurations in smart environments.

1.1 Contributions

In particular, this thesis makes these contributions:

- We design and implement an IoT-instrumented smart environment that leverages the capabilities of Digital Twin (DT) technology. By extending an open-source Software-Defined Network (SDN) controller into a Network Digital Twin Controller (NDTC), our system integrates multiple DT-based modules to support the creation and management of DTs. This extension not only facilitates real-time monitoring and control of physical IoT devices through their DTs but also enhances the flexibility and scalability of smart environment management.

- A major challenge in an NDTC is maintaining accurate and up-to-date synchronization between the states of Physical Twins (PTs) and their DTs. We address this challenge by formulating the state synchronization problem and proposing efficient algorithms: Optimal Update (OU) and Gradient-driven Update (GU). These algorithms dynamically adjust update frequencies and data granularities based on the available network bandwidth, ensuring that state deviations between PTs and DTs are minimized.

- Another challenge for an NDTC is the capability to perform what-if analysis, enabling administrators to predict the impact of various network configurations and operational scenarios on Quality of Service (QoS) metrics. We formulate the what-if analysis problem and develop an Optimal Selection (OS) algorithm, which intelligently selects a what-if analyzer to provide the most accurate QoS predictions within a given computing time budget. This capability is crucial for making decisions regarding network upgrades and reconfigurations in smart environments.

1.2 Limitations

In this thesis, we make the following assumptions:

- We assume that the bandwidth budgets for each PT, the NDTC, and the network topology are predefined. This simplification allows us to focus on optimizing our algorithms in a controlled setting, which reflects common real-world constraints.
- It is assumed that all networked devices are equipped with software to synchronize their states with the NDTC. This assumption simplifies integration and aligns with the trend of standardized protocols in modern IoT devices.
- We also assume that the switches in our network are compatible with the OpenFlow protocol, which supports flow and port statistics synchronization. While this may limit the applicability to non-OpenFlow environments, it ensures our research is based on a widely accepted and reliable standard.

1.3 Organization

In this thesis, we start by explaining our motivation in Chapter 1. Then, in Chapter 2, we describe the related techniques for IoT-instrumented smart environments. In Chapter 3, we review related works on building and optimizing NDTC. We presents our design of the NDTC and explains the function of each component in Chapter 4. In Chapter 5, we introduce the formulation of the two considered problems and our proposed algorithms. Chapter 6 describes the implementation of our testbed and the four what-if analyzers. Finally, in Chapter 7, we discuss the evaluations setup and results, and we conclude the thesis in Chapter 8.1.

Chapter 2

Background

In this chapter, we introduce the background knowledge of IoT, DTs, SDN, and the online machine learning techniques used in this thesis.

2.1 Internet-of-Things

The Internet of Things (IoT) refers to the concept where everyday physical objects are connected through embedded sensors, software, and other technologies, allowing them to communicate and share data over the Internet. This connectivity is essential for developing smart environments like cities, homes, and healthcare systems. The IoT depends on crucial technologies and diverse communication protocols that enable seamless interaction between devices, irrespective of their source or capabilities. These elements are vital for ensuring the scalability and reliability of IoT systems.

IoT architectures are designed to handle the vast amounts of data produced by connected devices. Integrating cloud computing and big data analytics is essential for processing this data, enabling real-time decision-making and enhancing efficiency across multiple sectors. However, the deployment of IoT comes with challenges, particularly concerning the security of communications and the privacy of user data. Addressing these challenges is critical to maximizing the benefits of IoT. Furthermore, establishing standardized protocols is necessary to ensure that devices from different manufacturers can work together seamlessly, facilitating the broad adoption of IoT technologies across various industries [1].

2.2 Smart Environments

Smart environments represent a significant advancement in urban development, combining IoT and big data technologies to create responsive, efficient, and sustainable urban spaces. Commonly referred to as smart cities, these environments leverage interconnected devices and advanced computing to streamline resource management, improve public services, and elevate the overall quality of life. By gathering and analyzing vast amounts of data, smart environments support informed decision-making and automation, which are crucial for managing the complex systems of modern cities [59].

Implementing smart environments brings challenges in areas like data management, security, and the need for scalable and adaptable infrastructures. To overcome these challenges, technologies such as fog computing are used to process data nearer to its source, reducing latency and enhancing real-time decision-making. This approach is especially crucial in urban settings where timely and accurate data processing is critical for services like traffic management, energy distribution, and public safety. Additionally, integrating big data analytics ensures efficient handling of the vast data generated, supporting smarter and more sustainable urban development [46].

2.3 Digital Twins

Digital Twins (DTs) is the emerging technologies that have been mainly applied to the manufacturing industry to facilitate the product life-cycle from design, development, validation, production, maintenance, and retirement. A DT is essentially a softwarized digital representation of an object, referred to as its Physical Twin (PT), where DT encompasses the data characterizing its PT's behavior. More comprehensive DT systems support bi-directional data updates between DTs and PTs, which enables applications like remote monitoring and control. Multiple DTs could interact with one another in the digital world, where diverse granularity levels of physical models are employed for *real-time* or *what-if* analysis through simulations. The simulation results, driven by real sensor data from the PTs, can be sent to PTs for autonomous actuation or human-in-the-loop decisions. In more generalized DT systems, humans and environments can also be abstracted away as DTs, although no concrete, single PT exists. In fact, some DTs can exist *without* their corresponding PTs, which further enable novel applications, such as testing an alternative or upgraded turbine design of an airplane engine, where the turbine may only exist in CAD (Computer-Aided Design) software. Different from traditional simulations, DTs without PTs can interact with other DTs, serving as virtual duplicates of real PTs, which can be physical objects, humans, or environments.

2.3.1 Features of Digital Twins

DT concepts can be applied to many applications, e.g., Savage [44] presented three sample DT applications: airplane manufacturing, building energy consumption, and Earth system. Rasheed et al. [39] gave an extensive survey on the value of DTs, enabling technologies, and common challenges. Furthermore, Minerva and Crespi [28] compiled an exhaustive list of possible DT features, which can be classified into four categories:

- **Object mirroring** features enable: (i) a DT to *represent* a PT, (ii) a DT to *reflect* the status of its corresponding PT, and (iii) a DT's status to be *entangled* with its PT's status in real-time and vice versa.

- **Object relation** features allow: (i) primitive DTs to be *aggregated* into more comprehensive ones, similar to their PT counterparts and (ii) DTs to be *augmented* for new or different characteristics than their PTs.

- **Data availability** features offer: (i) *persistent* data even when PTs are temporarily offline, (ii) *memorized* data of PTs in the past or different contexts, and (iii) *predicted* data of PTs in the future or under different contexts via DTs.

- **User interaction** features ensure that each DT is: (i) *managed* for timely execution without excessive resource consumption and (ii) *servitized* for on-demand accessibility to potentially many users.

Fig. 2.1 illustrates these ten features, which are built upon IoT, big data, ML, physical models, and service computing technologies.

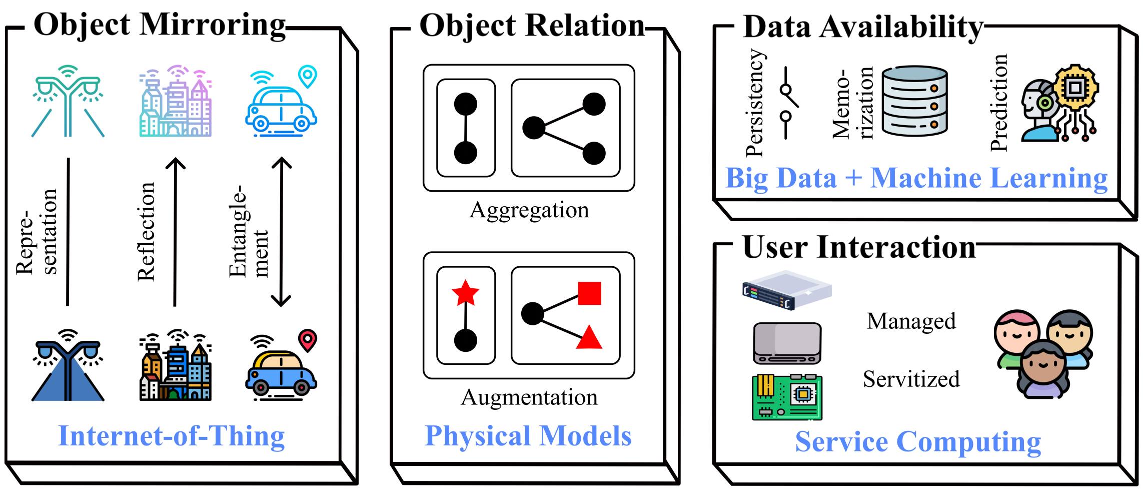


Figure 2.1: Unique features provided by DTs.

2.3.2 Popular Digital Twins Usage Scenarios

The majority of existing DT applications only leverage a subset of the abovementioned features for remote monitoring, controlling, predicting, and optimizing a system of PTs for better decision-making throughout their life cycle. Hence, this feature list enables us to come up with innovative usage scenarios and applications of DTs. Different research communities have considered different sets of DT features and tackled diverse research challenges. For example, Kim et al. [49] compared DTs with traditional simulation approaches, and then discussed unique challenges brought by DT systems. Major et al. [25] focused on the visualization aspect and demonstrated a graphic DT system for a variety of smart city applications, such as building information systems, energy management, weather forecasts, air quality monitoring, etc. El Saddik et al. [8] took a multimedia networking approach by considering multi-modal interactions, such as Extended Reality (XR), haptic, and gesture. They also pointed out the importance of Quality-of-Experience (QoE) driven communications and cybersecurity.

2.3.3 Urban Digital Twins Infrastructure

Only recently, DTs have been gradually applied to smart city applications. For instance, Mylonas et al. [29] pointed out the main difference between applying DTs in smart manufacturing and smart cities. They also presented sample smart city applications that will benefit from DTs. A wide spectrum of open challenges are listed, including the following ones that are more relevant to networked sensing systems in urban setup: (i) *privacy and security* concerns, which were well recognized in smart cities, (ii) *distributed and edge analytics* for city-wide services to citizens, (iii) *low latency services* for on-demand requests with stringent time requirements, (iv) *5G cellular technologies* for multi-tenant and diverse Quality-of-Service (QoS) supports, and (v) *scalability* by trading the complexity and accuracy of simulation models.

Several studies [26,31,37] tackled these challenges. For example, El Marai et al. [26] created DTs of road segments using IoT devices equipped with webcams, GPS readers, thermometers, and hydrometers. The challenges addressed in their paper were, however, closer to those in traditional IoT systems, including hardware deployment, detection accuracy, information visualization, and data security. Raes et al. [37] presented a framework for data exchanges among DTs and PTs in smart cities. The proposed framework has been tested for smart city applications related to traffic, air quality, and noise emission in three European cities. Nguyen et al. [31] studied how DTs can be used to accelerate the deployment and optimization of 5G cellular networks. In addition to 5G automotive and new radio emulations, DTs also enable continuous validation and optimization of new (alternative) 5G devices. Fig. 2.2 shows our layering structure DTs in smart environments.

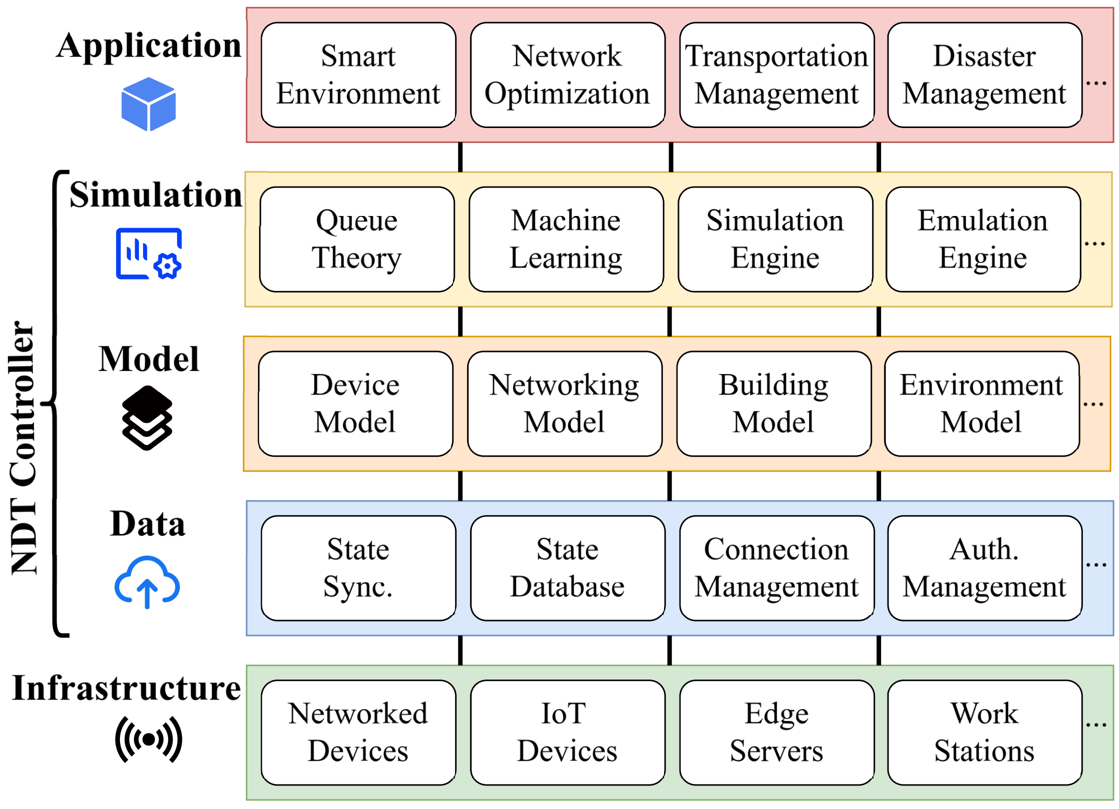


Figure 2.2: The layering structure of DTs in smart environments.

2.3.4 Network Digital Twins

NDTs have been studied in the literature, e.g., Rosello et al. [41] conducted a case study demonstrating how NDTs help configure and deploy industrial 5G networks. Li et al. [21] proposed solutions to enhance user satisfaction by optimally placing the DTs of networked servers. Lin et al. [22] developed NDTs for identifying key influencing factors of network behaviors. Guemes-Palau et al. [12] presented a technique to accelerate deep reinforcement learning for optimizing NDTs, significantly reducing the training time. Liu et al. [24] proposed NDTs for predicting resource demands and migrating Virtual Network Functions (VNFs), in order to minimize energy consumption and improve load balancing of wireless networks. Dai et al. [6] leveraged NDTs to optimize the computation offloading and resource allocation. These previous works [6, 12, 21, 22, 24, 41] focus on the high-level design of NDTs rather than their implementation and optimization. In contrast, our work proposes an NDTC that supports efficient synchronization and multiple what-if analyzers, enabling QoS prediction and optimization.

2.4 Software Defined Networking

Software-Defined Networking (SDN) Offers an innovative approach to network management by separating the control plane from the data plane, allowing for more flexible, programmable, and centralized control over network resources. This architecture streamlines network management by allowing administrators to modify network configurations and services through software-based controllers, abstracting the complexity of the underlying hardware. This technology enhances the network's adaptability to shifting demands and improves its scalability and efficiency, particularly in settings such as data centers and IoT networks [19]. The centralized structure of SDN is particularly effective for the rapid deployment of new services and the streamlined management of extensive, diverse network infrastructures [60].

Nevertheless, integrating SDN into environments with diverse devices and protocols, such as IoT, introduces significant challenges. The resulting heterogeneity in network technologies and devices requires careful management of QoS and security concerns. Recent research has suggested strategies to address these challenges by grouping heterogeneous controllers into homogeneous sets, which can significantly reduce response times and improve both QoS and security across the network [45].

2.5 Online Machine Learning

Online Machine Learning (OML) is designed for environments where data is continuously generated, requiring models to adapt in real-time. Unlike traditional machine learning methods that depend on static datasets, OML enables models to be dynamically updated as new data becomes available, making it particularly effective in situations that demand constant adaptation and learning. This approach is especially important in networking, where traffic patterns, user behavior, and other variables can shift rapidly, necessitating real-time adjustments to maintain optimal performance [13].

OML excels in managing changes in data patterns over time. By incrementally updating the model, OML can sustain its predictive accuracy without needing a complete retraining process. In networking, OML is well-suited for tasks like traffic prediction, anomaly detection, and resource allocation, where traditional batch learning methods may fall short in adapting to the dynamic nature of the environment. Recent research has underscored the benefits of OML in enhancing the adaptability and responsiveness of machine learning models in complex and rapidly evolving network scenarios [40].

Chapter 3

Related work

In this chapter, we review existing research on the development and optimization of NDTCs, state synchronization methods, and what-if analysis in networked environments.

3.1 Network Digital Twin Controller

In NDTs [3, 16], the NDTC builds a DT for each networked device, enabling innovative network management, monitoring, and optimization. Instead of developing NDTCs from scratch, researchers have extended SDN controllers for NDTs [27, 35, 51] to avoid reinventing the wheel. For instance, Raj et al. [38] developed an NDTC using knowledge graphs to improve NDTs' query efficiency. Hong et al. [14] built an NDTC that provides a unified interface to create DTs of heterogeneous devices. These works [14, 38], however, didn't optimize their NDTCs, setting them quite different from our current work. In fact, to the best of our knowledge, our thesis is the first attempt to optimize NDTCs in IoT-instrumented smart environments. In particular, we strive to: (i) minimize the state deviation between the DTs and PTs given network bandwidth budgets and (ii) maximize the prediction accuracy of heterogeneous what-if analyzers under a server computing time budget.

3.2 State Synchronization

Several previous studies have considered adaptive SDN state updates. For instance, Tian et al. [48] considered the trade-off between the state update frequency and bandwidth consumption. Poularakis et al. [36] optimized the update frequency among distributed SDN controllers for better overall performance. Nabil et al. [30] studied the data aggregation strategies, revealing the trade-offs among data granularity, reliability, and delay. Chen et al. [5] constructed DTs with different accuracy levels, update frequencies, and available bandwidth. Alanezi and Mishra [2] proposed edge-based NDTs equipped with prioritized synchronization mechanisms. None of the previous studies [2,5,30,36,48] exercised heterogeneous data granularities, while our proposed NDTC includes state synchronization algorithms to adapt to the bandwidth dynamics.

3.3 What-if Analysis

DT-enabled what-if analysis has been investigated in the past, e.g., Polverini et al. [34] proposed an NDT platform hosting DTs of the control plane to support what-if analysis in simulators to optimize the Border Gateway Protocol (BGP) configurations. Wieme et al. [55] implemented NDTs of a Bluetooth network using simulated network traffic to optimize network configurations. Wang et al. [52] proposed NDTs for 5G core networks to predict end-to-end QoS metrics using Graph Neural Networks (GNNs). Similarly, Ferriol-Galmes et al. [9] and Rusek et al. [42] applied GNNs to extract relations among the queuing policies, network topologies, routing algorithms, and ingested traffic patterns, aiming to predict network performance and optimize network configurations. Yu et al. [58] proposed an NDT platform utilizing GNNs to predict network performance and detect anomalies, enabling proactive service redeployment to ensure service stability and reduce delays in 6G edge networks. These works [9,34,42,52,55,58] built NDTs for homogeneous what-if analyzers. In contrast, our NDTC supports multiple what-if analyzers with varying accuracy and resource demands. The selections of what-if analyzers are made by our what-if analysis algorithm.

Chapter 4

Network Digital Twin Controller

In this chapter, we introduce our NDTC, which extends an SDN controller to manage and optimize networked environments using DT technology. We detail the core components and functionalities that enable real-time state synchronization and what-if analysis for smart environments.

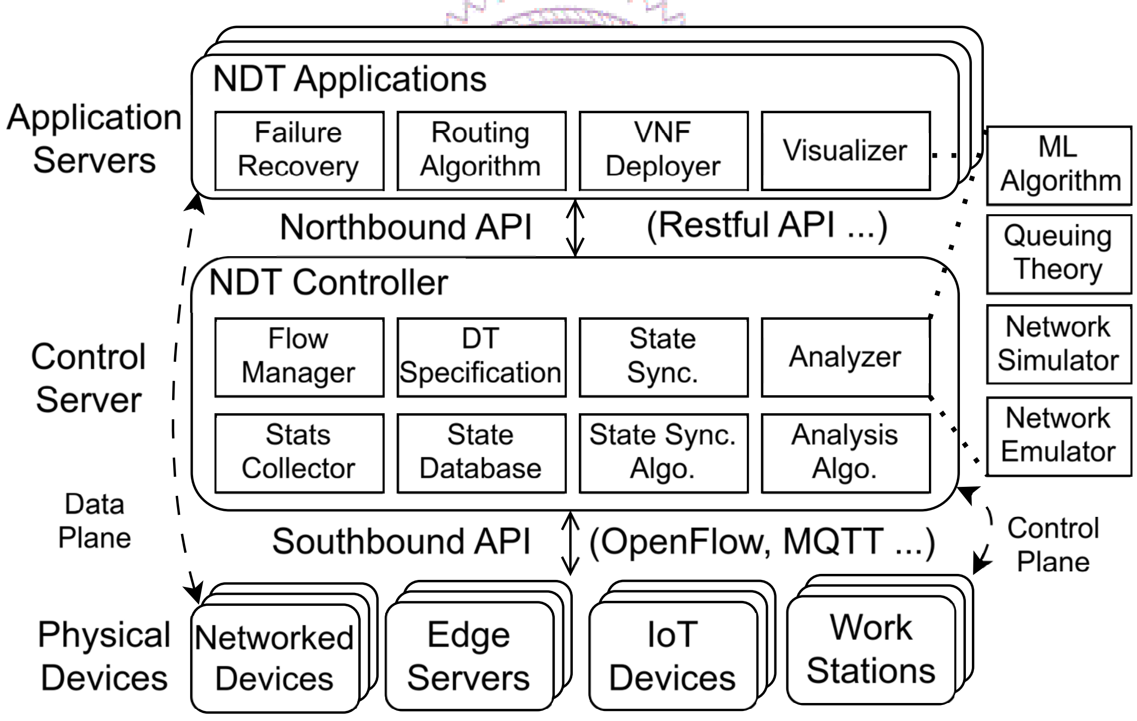


Figure 4.1: Overview of our NDTC.

4.1 System Overview

Fig. 4.1 shows our NDTC design. The NDTC is built on a commodity SDN controller that includes two key components: the flow manager and the stats collector. The flow manager maintains the flow tables of individual networked devices to create end-to-end data flows, while the stats collector gathers QoS metrics from these devices. Both components use the OpenFlow protocol to establish connections with switches and receive messages from them.

Our proposed NDTC offers a southbound API for physical devices to update their states and receive dictated actions, as well as a northbound API to exchange data with NDT applications. These NDT applications, such as failure recovery, routing algorithms, VNF deployment, and visualization, can be implemented in smart environments using real-time and historical data from DTs. For instance, the VNF deployer can leverage CPU and RAM usage data from all DTs to generate an optimized deployment plan for VNFs on the corresponding PTs.

To extend this SDN controller into an NDTC, we introduce six additional components in the following section: (i) DT specification, (ii) state database, (iii) state synchronizer, (iv) state synchronization algorithm, (v) what-if analyzer, and (vi) what-if analysis algorithm.

4.2 Components

- The DT specification defines the behavior of DTs, including the attributes and states of their corresponding PTs. It outlines the types and numbers of states, along with specific details of the PTs. These specifications are then used to create the DTs for the actual PTs.
- The state database stores the historical states of DTs for future analysis.
- The state synchronizer enables PTs to update their states. It supports multiple protocols, such as OpenFlow, MQTT, and NETCONF, allowing various types of physical devices to update their states.
- The state synchronization algorithm adaptively optimizes update frequency and data granularity based on the available network bandwidth budgets.
- The what-if analyzers are implemented using various approaches, such as ML algorithms, queuing theory, network simulators, and network emulators. Each what-if analyzer takes the network topology and traffic as input and outputs the predicted QoS metrics. However, the prediction results from different what-if analyzers vary in accuracy and differ from the ground truth, which is referred to as prediction error.
- The what-if analysis algorithm considers the trade-off between prediction error and computing time of the what-if analyzers. When users send a what-if analysis query to the NDTC, the algorithm selects the most suitable what-if analyzer.

Chapter 5

Optimization Problems and Solutions

In this chapter, we formulate the state synchronization problem and what-if analysis problem and present the solution for these two problems. Table 5.1 gives all symbols used in the thesis.

5.1 State Synchronization

In a DT-enable IoT instrumented smart environment, each PT periodically updates its states with the NDTC, which associates the states with the corresponding DT. Higher update frequencies and data granularities result in lower state deviations between PT and DT but lead to more network traffic. Conversely, lower update frequencies and data granularities cause higher state deviations. Therefore, the update frequency and data granularity need to be carefully selected to minimize the state deviation under the bandwidth budgets.

We consider N PTs, where the number of states of PT $n \in \{1, 2, \dots, N\}$ is denoted by K_n. We write the k-th state of PT n in time slot t as $s_p(n, t, k)$, where $n \in \{1, 2, \dots, N\}$, $k \in \{1, 2, \dots, K_n\}$, and $t \in \{1, 2, \dots, T\}$. Here, T represents the time horizon of each state synchronization problem. Similarly, we use $s_d(n, t, k)$ to represent the k-th state of DT n in time slot t. In addition, we let $m_{n,k}$ be the update message size of PT/DT n's state k. We write the update frequency $f_n \in \mathbb{R}^+$ to denote how many times PT n synchronizes the states with its DT per second. We sort PT n's K_n states in the descending priority and define PT n's data granularity $z_n \in \{1, 2, \dots, K_n\}$ to indicate how many states are synchronized in an update message. The total size of each update message of PT n at data granularity z_n is $\sum_{k=1}^{z_n} m_{n,k}$.

In terms of network bandwidth budgets, each PT n has bandwidth B_n, and the NDTC has a bandwidth B, shared by all PTs. We define the state deviation between PT/DT n at time slot t as $\theta(n, t)$, inspired by the definition in Vaezi et al. [50]. Each state k of DT n has a different importance, denoted as $\alpha_{n,k}$, where $0 < \alpha_{n,k} \leq 1$ and $\sum_{k=1}^{K_n} \alpha_{n,k} = 1, \forall n \in$

Table 5.1: Symbols Used in the thesis

Symbol	**Description**
N	Number of PTs
K_n	Number of States of PT n
T	Time horizon of each state synchronization problem
$s_p(n,t,k)$	k-th state of PT n in time slot t
$s_d(n,t,k)$	k-th state of DT n in time slot t
$m_{n,k}$	Update message size of PT/DT n's state k
f_n	Update frequency of PT n
z_n	Data granularity of PT n
B_n	Bandwidth of PT n
B	Bandwidth of the NDTC
$\theta(n,t)$	State deviation between PT/DT n at time slot t
$\alpha_{n,k}$	Importance weight of state k of DT n
β	Weight of EWMA
$\tilde{\theta}(f_n, z_n)$	Prediction model for PT n
∇f_n	Gradient of update frequency of PT n
∇z_n	Gradient of data granularity of PT n
Δf	System parameter of update frequency
Δz	System parameter of data granularity
Q	Number of queries
W	Number of what-if analyzers
E	Number of considered QoS metrics
C	Computing time budget
$e(q,w)$	Prediction error answering query q with what-if analyzer w
$c(q,w)$	Computing time answering query q with what-if analyzer w
p_e	Ground truth of QoS metric e
$\tilde{p}_e$	Predicted value of QoS metric e
$x_{q,w}$	Decision variable of query q using what-if analyzer w

$\{1, 2, \ldots, N\}$. We employ Exponentially Weighted Moving Average (EWMA) [7] with a weight $0 < \beta < 1$ to filter out high-frequency noise. $\alpha_{n,k}$ and β can be specified by the user, taking into account the significance of each state and the importance of previous state deviations. With these symbols, we write $\theta(n, t)$ as:

$$\beta \sum_{k=1}^{K_n} \alpha_{n,k} \frac{|s_d(n,t,k) - s_p(n,t,k)|}{s_p(n,t,k)} + (1-\beta)\theta(n, t-1). \tag{5.1}$$

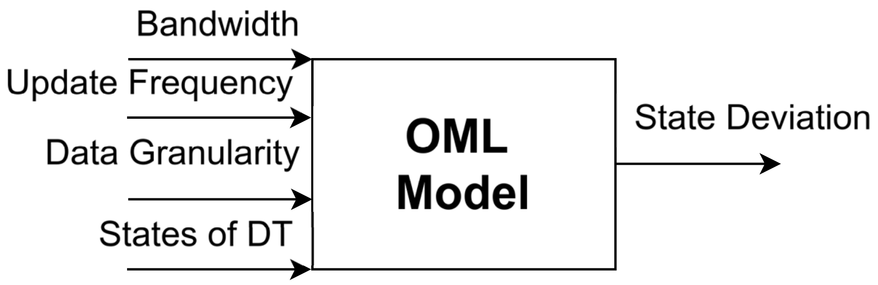


Figure 5.1: Overview of the OML model in our NDTC.

The actual values of $\theta(n, t)$ can be calculated whenever the NDTC receives an update message from PT n. However, to solve the state synchronization problem, the expected state deviation under any f_n and z_n needs to be predicted. To cope with this, we employ Online Machine Learning (OML) algorithms [4, 13, 40] to build a prediction model $\tilde{\theta}(f_n, z_n)$ so that $\theta(n, t) \simeq \tilde{\theta}(f_n, z_n)$ for PT n in any future time slot t. In particular, upon receiving an update message, the computed $\theta(n, t)$ is sent into an OML algorithm to refine $\tilde{\theta}(f_n, z_n)$, which is in turn used to solve our state synchronization problem. Fig. 5.1 illustrates the input and output of our OML algorithm, which takes the bandwidth of PT, update frequency, data granularity, and states of DT as inputs to predict the expected state deviation of DT. Several OML algorithms have been proposed [4], and we empirically compare the representative ones in Sec. 7.1 using a real testbed for the best-performing one.

With the symbols defined above, we write the state synchronization problem as:

$$\underset{\{f_n, z_n | \forall n\}}{\text{minimize}} \sum_{n=1}^{N} \tilde{\theta}(f_n, z_n) \tag{5.2a}$$

$$\text{s.t.: } f_n \times \sum_{k=1}^{z_n} m_{n,k} \leq B_n, \forall n \in \{1, 2, \ldots, N\}; \tag{5.2b}$$

$$\sum_{n=1}^{N} f_n \times \sum_{k=1}^{z_n} m_{n,k} \leq B. \tag{5.2c}$$

This formulation is an Integer Programming (IP) problem, which can be solved by generic solvers, such as CPLEX [15] and GLPK [10]. We develop a CPLEX-based Optimal Update (OU) algorithm to solve this problem optimally.

Although the OU algorithm gives the optimal update frequency and data granularity, it may take too long to complete for larger problems. Hence, we also propose an efficient Gradient-driven Update (GU) algorithm to solve this problem, which iteratively adjusts f_n and z_n, so as to reduce the state deviation without overloading the network. Across these iterations, we keep track of best-known f_n and z_n. At the beginning of each iteration, we use $\tilde{\theta}(f_n, z_n)$ to compute the gradients ∇f_n and ∇z_n, which represent the state deviation differences per unit bandwidth change. When increasing f_n or z_n, we prefer bigger absolute gradients as they lead to more state deviation reduction. In contrast, when decreasing f_n or z_n, we prefer smaller gradients. We use system parameters Δf and Δz to denote the steps for incrementing (or decrementing, as explained later) f_n and z_n, respectively.

More specifically, the GU algorithm consists of: (i) per-PT and (ii) overall phases for constraints in Eqs. (5.2b) and (5.2c), respectively. In the per-PT phase, for each PT n, we let $f_n = f_0$ and $z_n = z_0$. In the following iterations, after calculating ∇f_n and ∇z_n, we update f_n or z_n by:

$$\begin{cases} f_n \leftarrow f_n + \Delta f & \text{if } |\nabla f_n| \geq |\nabla z_n|; \\ z_n \leftarrow z_n + \Delta z & \text{if } |\nabla f_n| < |\nabla z_n|, \end{cases} \tag{5.3}$$

and move to the next iteration. By doing so, we invest bandwidth from B_n to the more rewarding option chosen between the update frequency and data granularity. This step is repeated until B_n is used up. The per-PT phase traverses through all N PTs. After that, we check if the bandwidth consumption of all PTs exceeds the bandwidth of NDTC, which is B. If it doesn't, the GU algorithm terminates and returns the f_n and z_n, $\forall n$.

Otherwise, the GU algorithm gets into the overall phase, in which it iteratively reduces the f_n or z_n of a selected PT n from $n \in \{1, 2, \ldots, N\}$, to minimize the state deviation increase per unit bandwidth change. At the beginning of each iteration, we calculate the gradients ∇f_n and ∇z_n for all PTs $n = 1, 2, \ldots, N$. We then select the smallest absolute gradient among all $2N$ of them. Assuming ∇f_{n^*} is the smallest one in its absolute value, we update f_{n^*} by:

$$f_{n^*} \leftarrow f_{n^*} - \Delta f, \tag{5.4a}$$

so that we cut some bandwidth usage while controlling the state deviation increase. On the other hand, if ∇z_{n^*} is the smallest gradient in its absolute value, we update z_{n^*} by:

$$z_{n^*} \leftarrow z_{n^*} - \Delta z. \tag{5.4b}$$

This is repeated until the total bandwidth consumption of all PTs no longer exceeds B.

The GU algorithm then returns the f_n and z_n, $\forall n$, computed in the overall phase.

To demonstrate that changes in update frequency and data granularity can reduce both the magnitude and the continuity of state deviation, we analyze the data collected in 7.1. As shown in Fig. 5.2, increasing the update frequency and data granularity not only decreases the overall state deviation but also significantly reduces its continuity, thereby confirming our hypothesis.

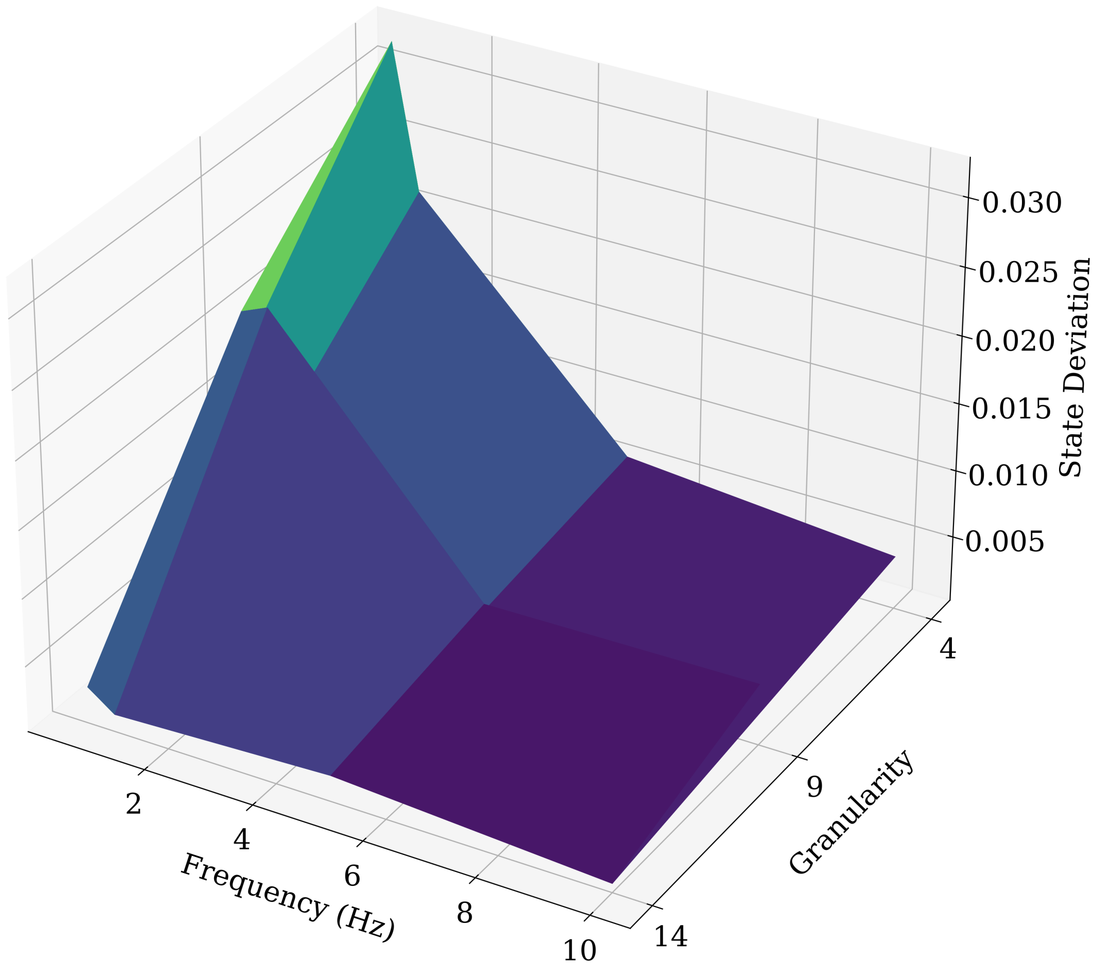


Figure 5.2: Impact of update frequency and data granularity on state deviation and its continuity.

5.2 What-if Analysis

We consider Q queries sent by the administration of a smart environment, where each query $q \in \{1, 2, \ldots, Q\}$ can be answered by one of the what-if analyzers $w \in \{1, 2, \ldots, W\}$. Here, W denotes the total number of heterogeneous what-if analyzers. The job of what-if analyzers is to predict the QoS metrics of individual networked devices in the smart environment. We let E be the number of considered QoS metrics and C be the comput-

Algorithm 1 Gradient-driven Update Algorithm (GU)

```
for $n \in \{1, 2, \ldots, N\}$ do
    $r = 0$
    while $r < MaxIteration$ do
        $f_n^{'} \leftarrow f_n$, $z_n^{'} \leftarrow z_n$
        $\nabla f, \nabla z \leftarrow$ ComputeGradient$(f_n, \Delta f, z_n, \Delta z)$
        if $|\nabla f| \geq |\nabla z|$ then
            $f_n^{'} \leftarrow f_n + \Delta f$
        else
            $z_n^{'} \leftarrow z_n + \Delta z$
        if $f_n^{'} \times \sum_{k=1}^{z_n^{'}} m_{n,k} > B$ then
            break
        $f_n \leftarrow f_n^{'}$, $z_n \leftarrow z_n^{'}$
        $r \leftarrow r + 1$
while $\sum_{n=1}^{N} f_n \times \sum_{k=1}^{z_n} m_{n,k} > B$ do
    $\mathbf{F} = \{\}, \mathbf{Z} = \{\}$
    for $n \in \{1, 2, \ldots, N\}$ do
        $\nabla f, \nabla z \leftarrow$ ComputeGradient$(f_n, \Delta f, z_n, \Delta z)$
        $\mathbf{F}$.append$(\nabla f)$, $\mathbf{Z}$.append$(\nabla z)$
    Let $\nabla f_{n^*}$ be the smallest absolute gradient in $\mathbf{F}$, and $\nabla z_{n^*}$ be the smallest absolute
        gradient in $\mathbf{Z}$
    if $|\nabla f_{n^*}| < |\nabla z_{n^*}|$ then
        $f_{n^*} \leftarrow f_{n^*} - \Delta f$
    else
        $z_{n^*} \leftarrow z_{n^*} - \Delta z$
Return $\{f_n, z_n | n \in \{1, 2, \ldots, N\}\}$
```

ing time budget for answering Q queries on the edge/cloud servers. Our what-if analysis problem is to select the most suitable what-if analyzer for each query to minimize the QoS prediction error without exceeding the computing time budget.

Note that different what-if analyzers achieve different QoS prediction errors and consume different computing times. Without loss of generality, we assume all W what-if analyzers are sorted in the ascending order of the computing time and also in the descending order of the QoS prediction error. We write the prediction error and computing time answering query $q \in \{1, 2, \ldots, Q\}$ with what-if analyzer $w \in \{1, 2, \ldots, W\}$ as $e(q, w)$ and $c(q, w)$, respectively. In practice, we execute each what-if analyzer w for query q on a given IoT network topology and traffic pattern multiple time and calculate the average prediction error as:

$$e(q, w) = \frac{1}{E} \sum_{e=1}^{E} \left\| \frac{p_e - \tilde{p}_e}{p_e} \right\|, \tag{5.5}$$

where p_e and $\tilde{p}_e$ denote the ground truth and predicted values of QoS metric e. Similarly, we set $c(q, w)$ to be the average computing time. Next, we let $x_{q,w} \in \{0, 1\}$, $\forall q \in \{1, 2, \ldots, Q\}$ and $w \in \{1, 2, \ldots, W\}$ be the decision variables. Here, $x_{q,w} = 1$ iff we answer query q using what-if analyzer w. It it not hard to see that $\sum_{w=1}^{W} x_{q,w} = 1, \forall q \in \{1, 2, \ldots, Q\}$.

With the notations developed above, we write the what-if analysis problem as:

$$\underset{\{x_{q,w} | \forall q, w\}}{\text{minimize}} \sum_{q=1}^{Q} \sum_{w=1}^{W} e(q, w) \times x_{q,w}$$

$$\text{s.t.: } \sum_{q=1}^{Q} \sum_{w=1}^{W} c(q, w) \times x_{q,w} \leq C.$$

This formulation is also an IP problem. We develop a CPLEX-based Optimal Selection (OS) algorithm to solve the what-if analysis problem optimally.

Chapter 6

Implementations

In this chapter, we describe the implementation of our NDTC on a real testbed, detailing the integration of six DT-related components into an open-source SDN controller. Additionally, we outline the setup of our testbed and the baseline algorithms used to evaluate the performance of our state synchronization and what-if analysis algorithms.

6.1 Testbed

We have implemented the proposed NDTC based on the open-source Ryu SDN controller [43] by adding six DT-related components to it. We have also implemented four what-if analyzers leveraging ML algorithm, queuing theory, network simulator, and network emulator. The ML algorithm analyzer employs a recent GNN [42]. With queuing theory, we model each network link as an M/M/1 queue using the SimPy library [33]. We adopt NS-3 [32] as our network simulator. For the network emulator, we integrate some data flows from our testbed with an NS-3 simulator. Fig. 6.1 illustrates the implementation of the network emulator. We connect the network interface of the application server to the Linux bridge and establish a Linux TAP, which serves as the Ethernet tunnel software network interface, also linked to the bridge. Subsequently, we connect the TAP to the corresponding network interface in the NS-3 simulation, enabling the transmission of network traffic from the real network interface to the virtual network interface. All these enhancements are done through four thousand lines of new Python code.

Fig. 6.2 depicts the workflow of our NDTC. When PT updates its states to the NDTC, the NDTC updates the corresponding DT's states and uses the previous states from the DT to calculate the actual state deviation. It then inputs the current update frequency, data granularity, states, and the actual state deviation of the DT into the OML model for online learning.

We built a real testbed for all experiments, as illustrated in Fig. 6.3. Also, Fig. 6.4

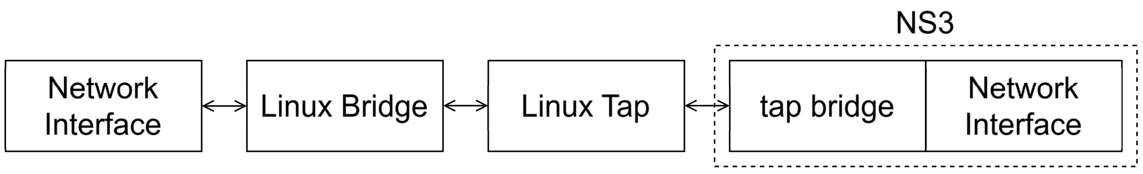


Figure 6.1: Overview of the emulator.

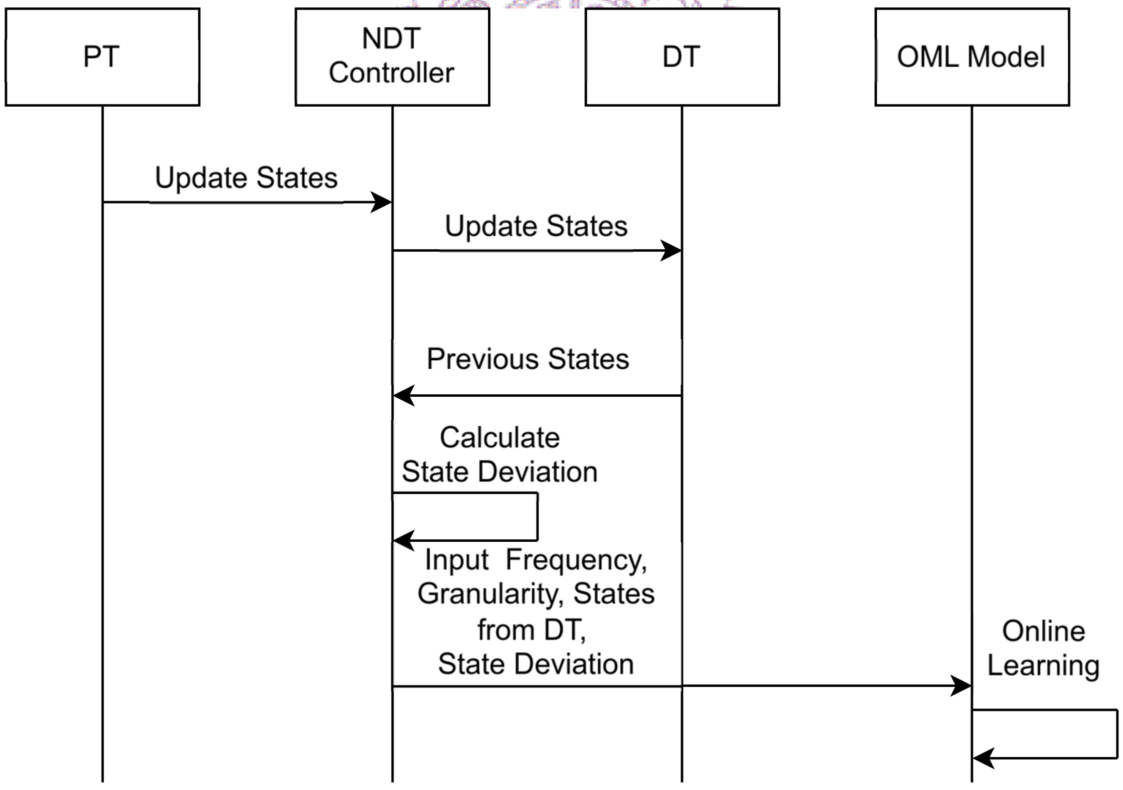

PT
NDT Controller
DT
OML Model
Update States
Update States
Previous States
Calculate State Deviation
Input Frequency, Granularity, States from DT, State Deviation
Online Learning

Figure 6.2: Workflow of our NDTC.

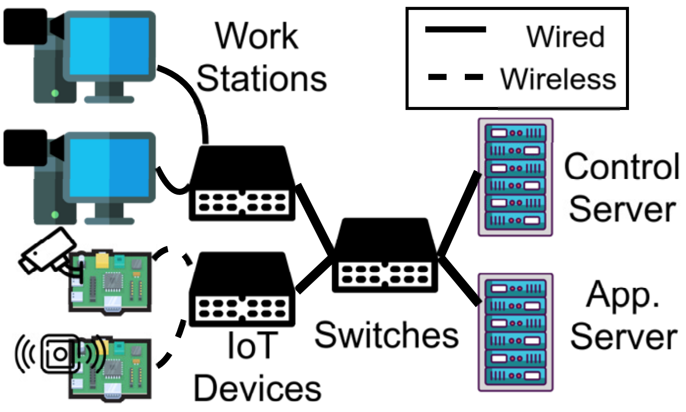


Figure 6.3: The real testbed for evaluations.

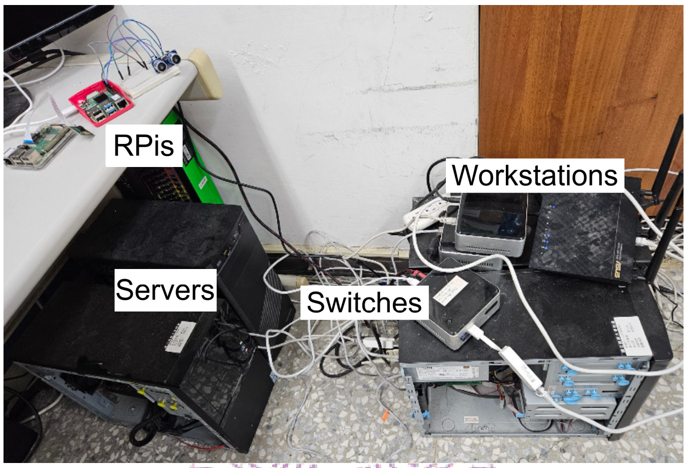


Figure 6.4: A snapshot of our testbed.

shows a snapshot of our testbed. The testbed consists of seven networked devices and two servers connected by wired and wireless networks. The network switches are based on the well-known OpenvSwitch [23]. Our proposed NDTC runs on a control server, while applications run on an applications server. We used a PC with an AMD Ryzen 5 CPU @ 3.6 GHz, 6 cores, and 64 GB RAM as the control server and a PC with an AMD Ryzen 5 CPU @ 3.6 GHz, 4 cores, and 16 GB RAM as the application server. Two classroom workstations stream two camera feeds of the blackboard and instructor to the application server, which in turn disseminates the lecture videos to online students over the Internet. The two IoT devices are based on Raspberry Pi (RPi), equipped with a camera and an ultrasound sensor, respectively. The video and distance readings are streamed to the application server for illegal parking detection and pedestrian counting, which trigger notifications to law enforcement and brightness changes of streetlights, respectively. If not otherwise specified, all videos are encoded in H.264 at 10 Mbps, and the time-series ultrasound distances occupy 0.5 kbps. These three smart-environment applications run throughout our experiments, producing realistic workloads.

6.2 Baseline Algorithms

The state synchronization and what-if analysis algorithms are the main software components under evaluation, which, to our best knowledge, were never considered in the literature [2, 3, 5, 14, 16, 27, 35, 38, 48, 51]. Hence, in addition to our proposed OU, GU, and OS algorithms, we also implemented four baseline algorithms for comparison. For the state synchronization problem, we propose two baseline algorithms:

- Alternative Update (AU) Algorithm: The AU algorithm alternates between randomly incrementing the update frequency or the data granularity for all PTs in the network until the bandwidth budget is exceeded. The AU algorithm ensures that both frequency and granularity are gradually increased, providing a straightforward mechanism to balance the load across the network. However, it does not account for the specific requirements or conditions of individual PTs, which may lead to suboptimal state synchronization under varying network conditions.

- Random Update (RU) Algorithm: The RU algorithm randomly selects either the update frequency or data granularity of a PT and increments it. This process is repeated for random PTs until the bandwidth budget is exceeded. The randomness in selection introduces variability in the synchronization process, which might help prevent the network from becoming too predictable and could handle diverse conditions better than a uniform approach. However, this method may also lead to inefficient use of bandwidth, as the updates are not driven by the actual needs of the PTs or the network.

For the what-if analysis problem, we also propose two baseline algorithms:

- Alternative Selection (AS) Algorithm: The AS algorithm consistently selects the most complex what-if analyzer available for each query, ensuring that the maximum level of detail and computational resources are used in the analysis. This approach assumes that more complex analyzers yield better predictions. The selection process continues until the computing time budget is close to being exceeded. While this method prioritizes accuracy and thoroughness, it may result in unnecessarily high computational costs, particularly when simpler analyzers could provide sufficient accuracy.

- Random Selection (RS) Algorithm: The RS algorithm randomly chooses a what-if analyzer for each query, continuing until the computing time budget is exceeded. The randomness allows the RS algorithm to explore a diverse set of analytical methods, potentially uncovering efficient solutions that might be overlooked by more deterministic approaches. However, like the RU algorithm, this randomness can lead to inefficiencies, as it does not account for the specific characteristics of the queries or the capabilities of the analyzers.

Chapter 7

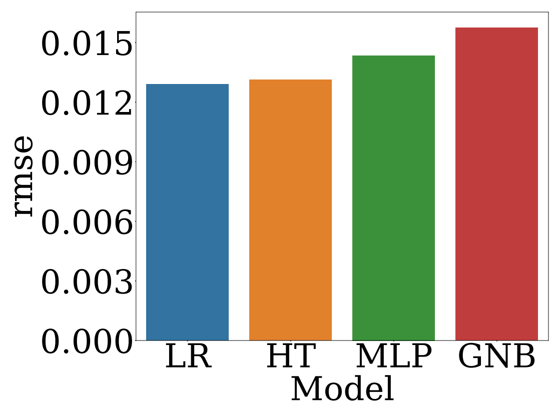
Evaluations

In this chapter, we present a comprehensive evaluation of our NDTC implementation, focusing on the effectiveness of our proposed state synchronization and what-if analysis algorithms. Through a series of experiments conducted on a real testbed, we analyze model deviations, assess the performance of baseline algorithms, and examine the overhead introduced by our solutions to validate their functionality and efficiency in smart environments.

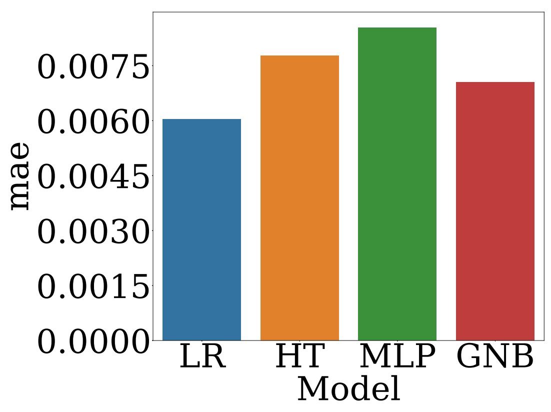
7.1 Model Deviation

With our testbed, we derive the models used by our proposed solution. For the state synchronization problem, we develop an OML-based state deviation model $\tilde{\theta}(\cdot)$, which takes the update frequency, data granularity, and DT states as inputs to predict the state deviation. We consider four representative OML algorithms: Linear Regression (LR), Hoeffding Tree (HT), Multi-layer Perceptron (MLP), and Gaussian Naive Bayes (GNB) [4]. We conduct experiments on our testbed to compare their performance in Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2).

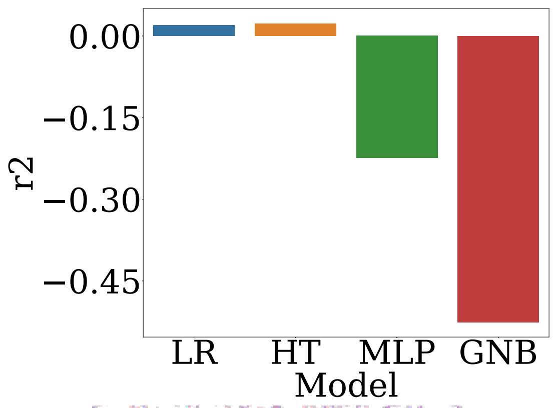
In particular, we measure the QoS metrics under random settings chosen from: (i) update frequency $\in \{0.1,0.5,1,3,10\}$ Hz, (ii) data granularity $\in \{4,9,14\}$, where the total number of states is 14, and (iii) bandwidth budget $\in \{0.5,1,2,4\}$ Mbps In total, 1000 state measurements are taken and randomly divided into 80/20 for training/testing. Our grid search gives the best hyperparameters for experiments, e.g.: (i) the learning rate of LR is 0.01, (ii) the number of leaves of HT is 50, and (iii) the number of the neurons of the hidden layer of MLP is 10. Fig. 7.1 shows the evaluation metrics of OML algorithms. We found that the LR algorithm constantly outperforms other OML algorithms by nontrivial margins: at least 4.3% in RMSE, 3.3% in MAE, and 35.5% in R^2 were observed. Hence, we employ the LR algorithm for the model $\tilde{\theta}(\cdot)$.



(a)



(b)



(c)

Figure 7.1: Evaluation metrics of OML algorithms: (a) RMSE, (b) MAE, and (c) R-Squared.

For the what-if analysis problem, unlike what-if analyzers based on the queuing theory, network simulator, and network emulator, the ML-based what-if analyzer needs to be trained with real measurement results from a site survey. On our testbed, we carry out 100 measurements under the workloads generated by the three smart-environment applications. We use 20 % of all the measurements for testing while the rest for training our ML-based what-if analyzer. We consider two QoS metrics: one-way delay and delay jitter in our discussion. Fig. 7.2 compares the trade-off between $e(\cdot)$ and $c(\cdot)$ among over what-if analyzers, which confirms our intuitions, e.g., the network emulator achieves the smallest prediction error but consumes the most computing time. The derived models are used in the rest of the evaluations.

7.2 Setup

We conduct experiments using our testbed to evaluate our NDTC solution, focusing on the two optimization components. We empirically choose the following system param-

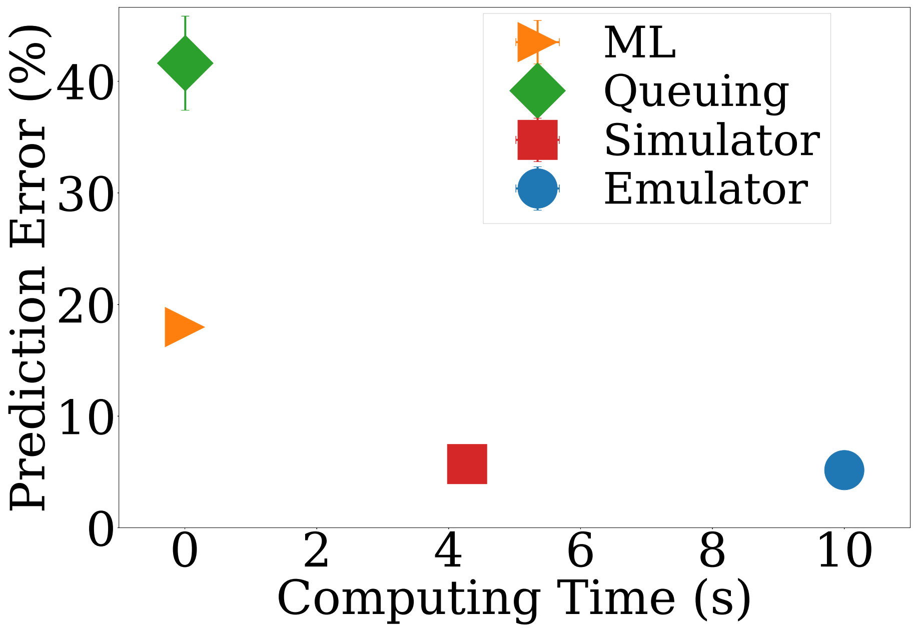


Figure 7.2: Prediction error and computing time of our what-if analyzers.

eters: (i) $f_0 = 1$ Hz, (ii) $z_0 = 4$, (iii) $\Delta f = 0.1$ Hz, (iv) $\Delta z = 1$, (v) $N = 7$, (vi) $T = 10$ s, (vii) $K_n = 14, \forall n$, (viii) $\alpha_{n,k} = 1/14, \forall n$, (ix) $\beta = 0.9$, and (x) $W = 4$. We vary the following parameters, where the default values are underlined: (i) control server bandwidth budget: $B \in \{5, 10, 15, \underline{30}\}$ Mbps, (ii) networked device bandwidth budget: $B_n \in \{0.5, 1, 2, \underline{4}\}$ Mbps, (iii) number of queries: $Q \in \{15, \underline{20}, 25, 30\}$, and (iv) computing time budget: $C \in \{60, \underline{120}, 240, 480\}$ seconds. For each combination of the parameters, we conduct experiments with different state synchronization and what-if analysis algorithms for comparison. For statistically meaningful results, we repeat each experiment for 10 runs, where each run lasts for 10 seconds. For each what-if analysis query, we randomly select a networked device with a QoS metric to mimic an inquiry made by the administration.

We measure the following metrics:

- State deviation: $\theta(\cdot)$ is computed whenever a state update message is received, which is the objective function in Eq. (5.1).
- Prediction error: $e(\cdot)$ of the queries from the administration, which is the objective function in Eq. (5.6).
- Running time of the optimization algorithms.
- Memory utilization of the application server.
- CPU utilization of the application server.

- Control message throughput at the control server.

We report the average of measured results with 95% confidence intervals whenever applicable.

7.3 Results

The proposed NDTC functions correctly without incurring excessive overhead. To validate the functionality of our NDTC, we zoom into a sample run of our experiments with default parameters. Fig. 7.3 plots PT/DT states from sample networked devices. This figure shows that the states of DT are updated every second, which is the employed update frequency. Table 7.1 gives the QoS prediction and computing time of individual what-if analyzers for a sample what-if question: ”What are the expected QoS measurements if we reduce the bitrate of all video streams from 10 to 5 Mbps?” This table depicts that all our what-if analyzers meet the functional requirements: QoS predictions are provided. Fig. 7.4 presents the control message throughput at the control server. This figure reveals that the overhead is manageable: about 200 (150) kbps in the RX (TX) directions even when the update frequency is as high as 8 Hz. *In summary, these sample results confirm that our NDTC functions well with an acceptable control message overhead.*

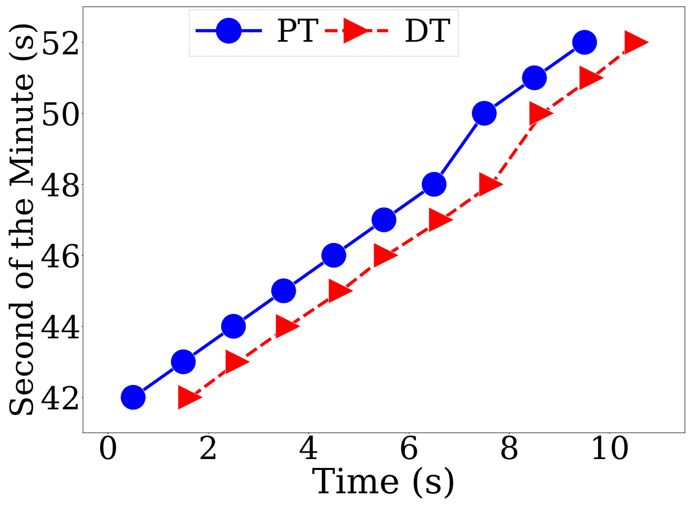
Our proposed OU and GU algorithms reduce the state deviation of networked devices. Fig. 7.5 presents the state deviation achieved by different state synchronization algorithms under the default parameters. Figs. 7.5(a) and 7.5(b) report the expected state deviation from our optimization problem from a sample network switch and across all networked devices, respectively. These two figures reveal that our proposed OU algorithm indeed leads to the lowest state deviation, up to 8.39 times smaller than the baseline algorithms. Moreover, our proposed GU algorithm also results in a smaller state deviation than the baseline algorithm, although the gap is small. Figs. 7.5(c) and 7.5(d) give the actual state deviation collected from our testbed. These two figures show that our proposed GU algorithm significantly outperforms the baseline algorithms by a large margin, up to a 98.92% reduction in the state deviation is observed. Also, Our proposed OU algorithm also outperforms the baseline algorithms by up to 99.49% in state deviation. We note that the GU algorithm performs better in actual state deviation compared to expected state deviation. In fact, the GU algorithm even outperforms the OU algorithm in terms of actual state deviation in our testbed, as shown in Fig. 7.5(d). Moreover, Figs. 7.7-7.12 show the result under different networked device bandwidth and control server bandwidth. Our GU and OU algorithms also outperform in expected state deviation and actual state deviation. *In summary, our proposed OU and GU algorithms successfully reduce the expected and actual state deviation in the optimization formulation and real testbed.*

Our proposed algorithms incur acceptable overhead. Fig. 7.6 presents the overall overhead across all runs under the default parameters. Fig. 7.6(a) reveals that the GU algorithm runs much faster than the OU algorithm, with a 10.46 times speed-up in terms of the running time. Furthermore, even for the optimal OU algorithm, its running time is merely $\sim$ 800 ms. Considering the state synchronization problem only needs to be solved whenever the environment (like the control server bandwidth) is changed, sub-second running time is quite reasonable. Figs. 7.6(b)–7.6(d) give the control message throughput, application server memory utilization, and application server CPU utilization, respectively. Fig. 7.6(b) depicts that our proposed algorithms increase the control message throughput: up to 40.19% and 132.97% increases are observed. Even with such an increase, the proposed GU algorithm only consumes $\sim$ 400 kbps of total bandwidth, which is not high in modern networks. Figs. 7.6(c) and 7.6(d) confirm that the state synchronization algorithms impose no impact on the application server overhead.

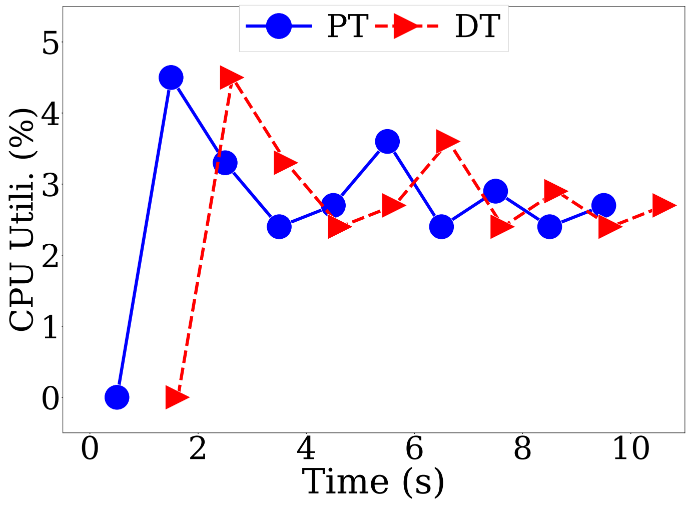
In summary, our proposed OU algorithm incurs higher control message throughput. With that said the various aspects of the overheads are all acceptable.

Our proposed OS algorithm significantly reduces the prediction error and scales to larger what-if analysis problems. Fig. 7.13 compares our proposed OS algorithm against two baseline algorithms under different numbers of queries. Fig. 7.13(a) shows that our optimal OS algorithm always leads to the lowest prediction error: up to 107.55% and 103.35% reductions are observed, compared to the AS and RS algorithms, respectively. Fig. 7.13(b) reveals that unlike the AS and RS algorithms, which demonstrate increasing running time with more queries, our proposed OS algorithm incurs almost constant running time, up to 101.59 ms. Fig. 7.14 compares our proposed OS algorithm against two baseline algorithms under different computing time budgets. Fig. 7.14(a) shows that our optimal OS algorithm always achieves the lowest prediction error: up to 75.77% and 83.63% reductions are observed, compared to the AS and RS algorithms, respectively. Fig. 7.14(b) shows that the OS algorithm terminates faster when the computing time budget is looser, which is intuitive. More importantly, this figure confirms that even with a challenging 60-s computing time budget, our OS algorithm takes no more than 70.67 ms to terminate.

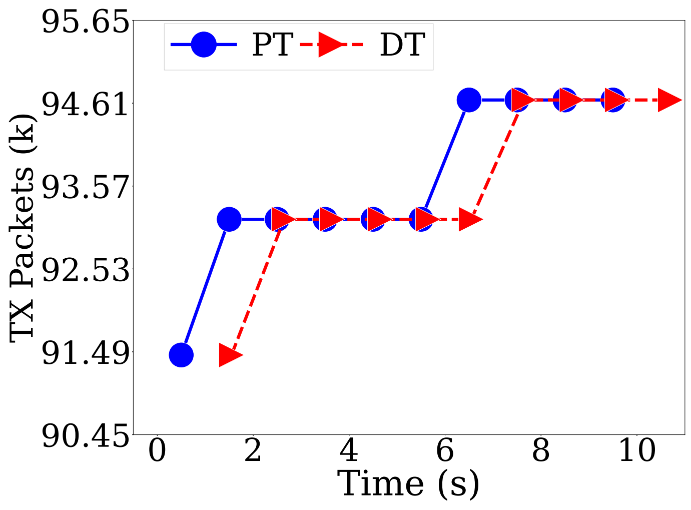
In summary, our proposed OS algorithm delivers the smallest prediction error for all what-if analysis queries yet terminates within 1 s.



(a)

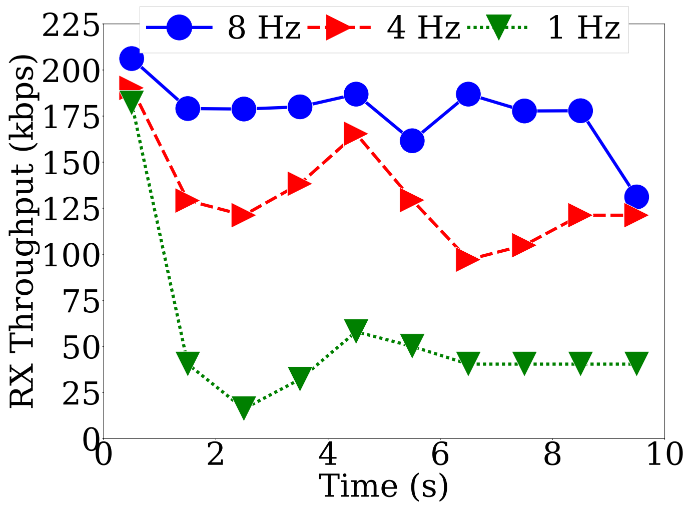


(b)

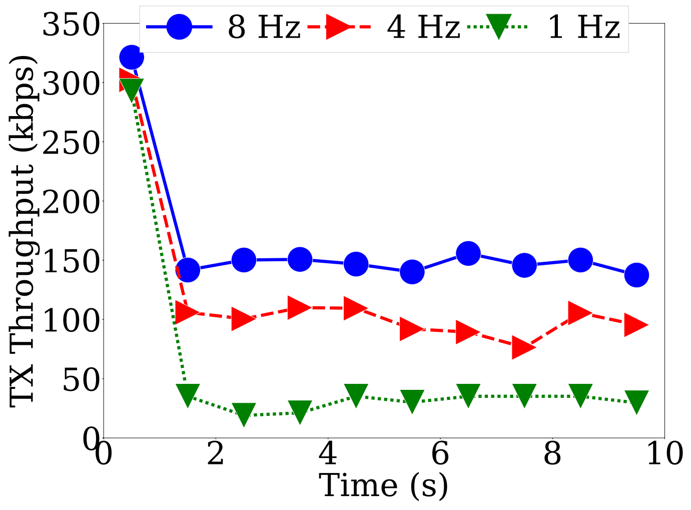


(c)

Figure 7.3: Comparison of PT/DT states from a sample run at: (a) an IoT device, (b) a workstation, and (c) a network switch.



(a)

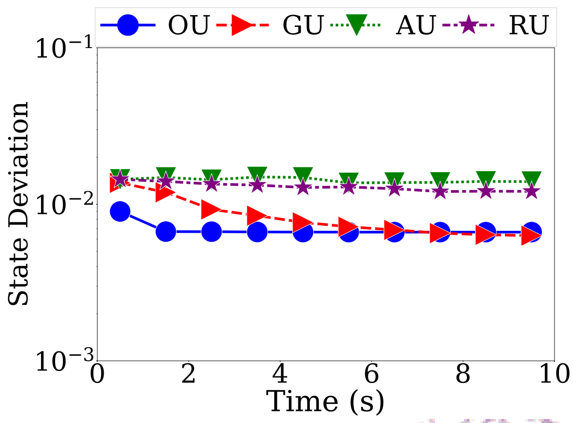


(b)

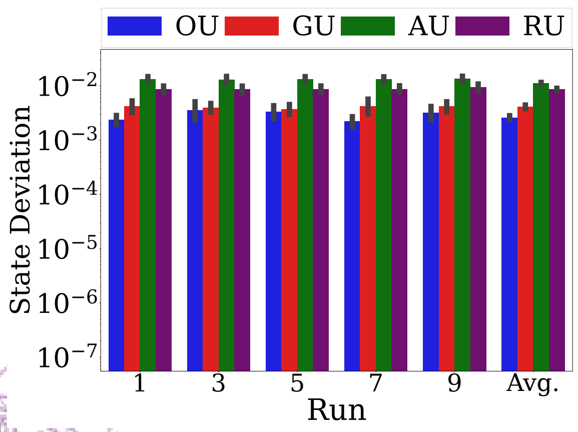
Figure 7.4: Control server throughput under different update frequencies: (a) RX and (b) TX.

Table 7.1: Sample Prediction and Computing Time of Different What-if Analyzers

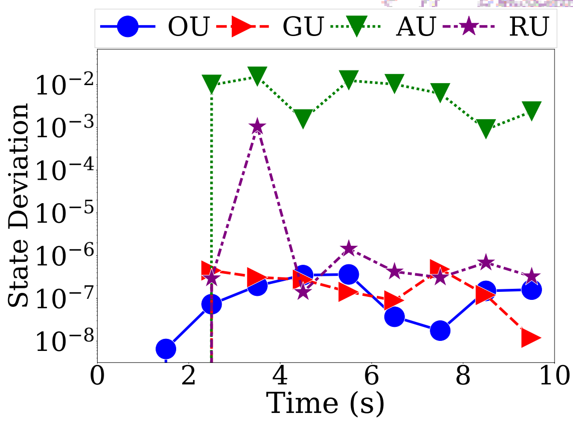
Metrics	**Pred. Delay (ms)**	**Pred. Jitter (ms)**	**Comp. Time (ms)**
ML	19.07	16.88	2.38
Queuing	43.50	40.40	5.16
Simulator	10.32	0.34	4338
Emulator	11.06	0.10	10,000
Ground Truth	0.70	0.15	10,000



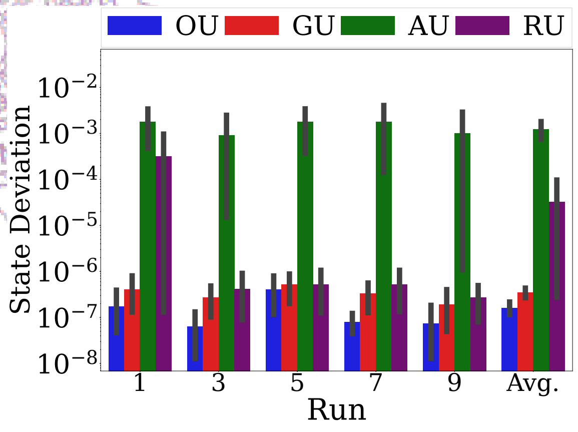
(a)



(b)

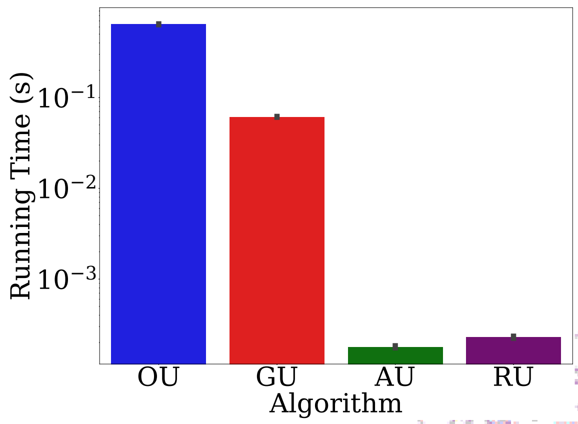


(c)

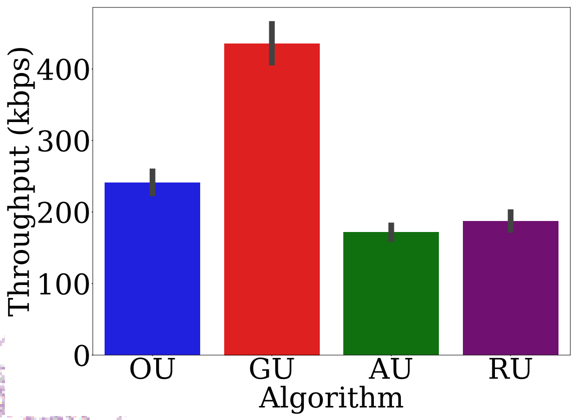


(d)

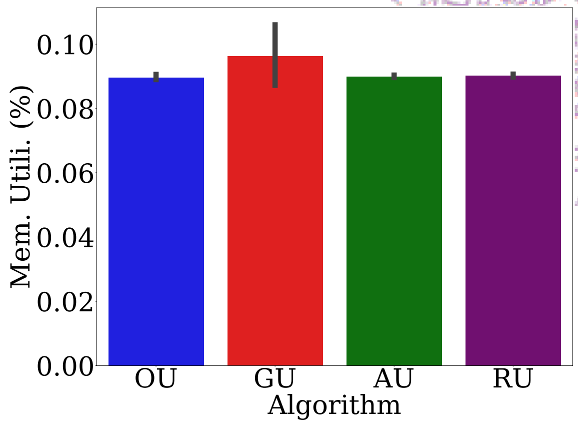
Figure 7.5: State deviation from different state synchronization algorithms. Expected state deviation: (a) from a network switch in a sample run and (b) across all networked devices. Actual state deviation: (c) from a network switch in a sample run and (d) across all networked devices.



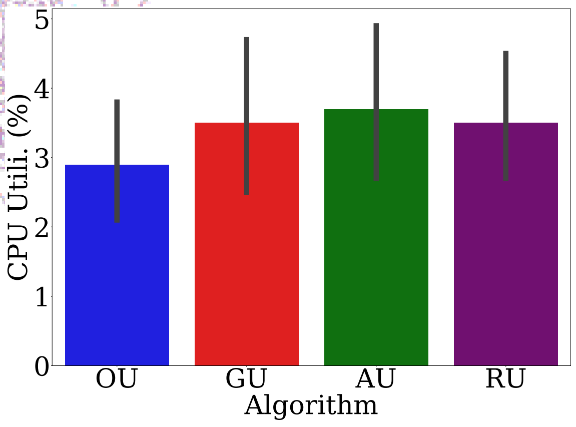
(a)



(b)

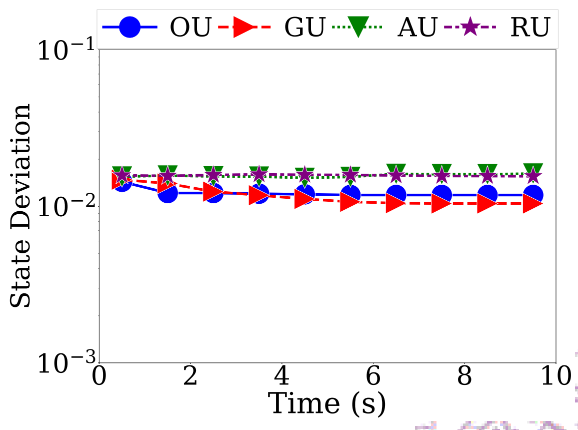


(c)

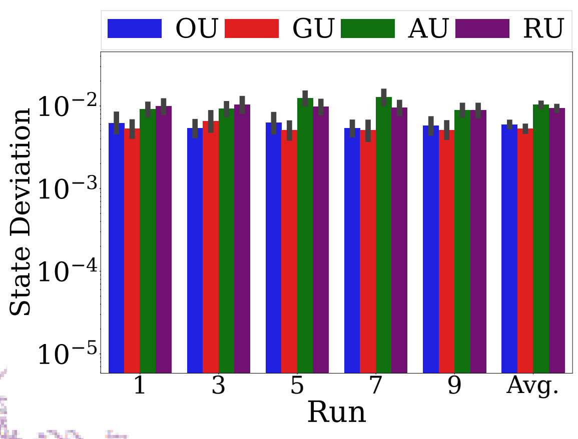


(d)

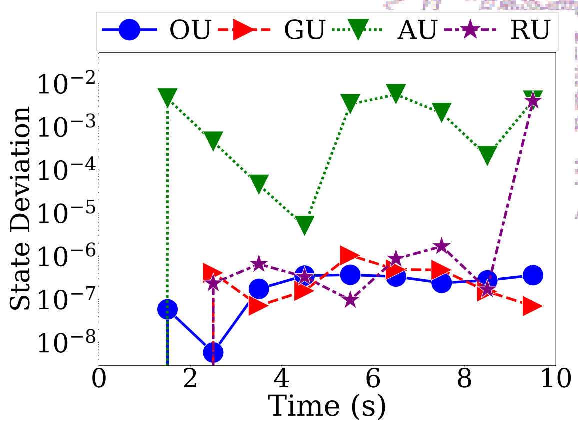
Figure 7.6: Overhead caused by different state synchronization algorithms: (a) running time, (b) total control server throughput, (c) application server CPU utilization, and (d) application server memory utilization.



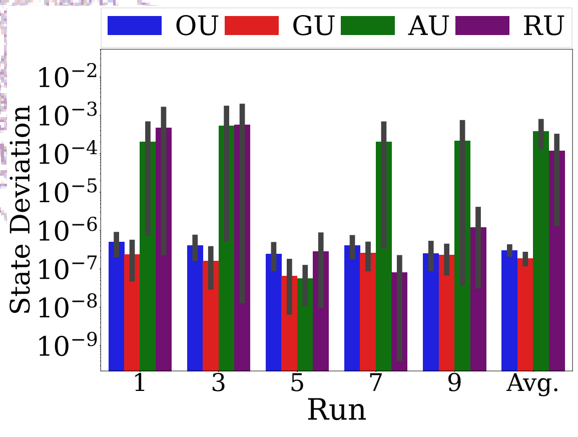
(a)



(b)



(c)



(d)

Figure 7.7: State deviation from different state synchronization algorithms under 2 Mbps networked device bandwidth. Expected state deviation: (a) from a network switch in a sample run and (b) across all networked devices. Actual state deviation: (c) from a network switch in a sample run and (d) across all networked devices.

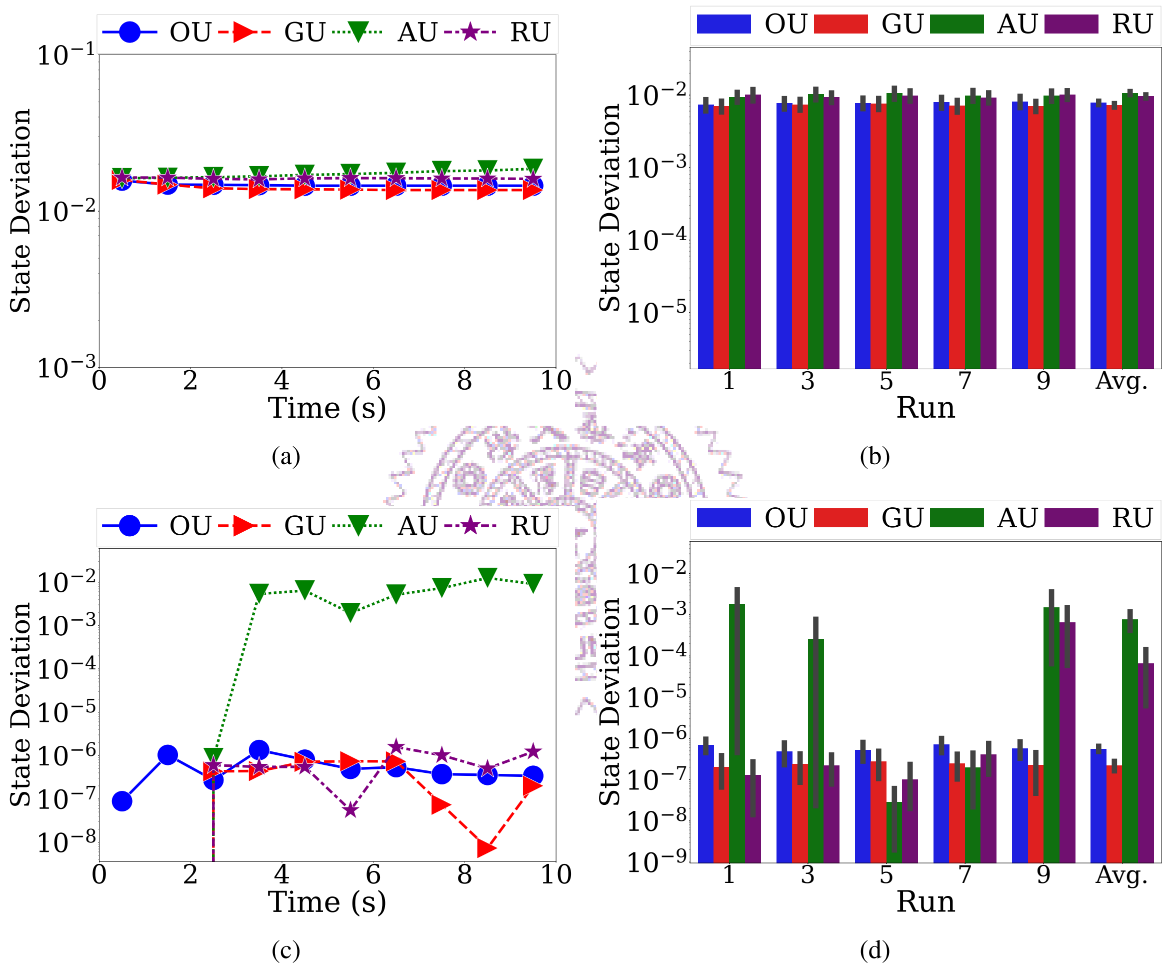
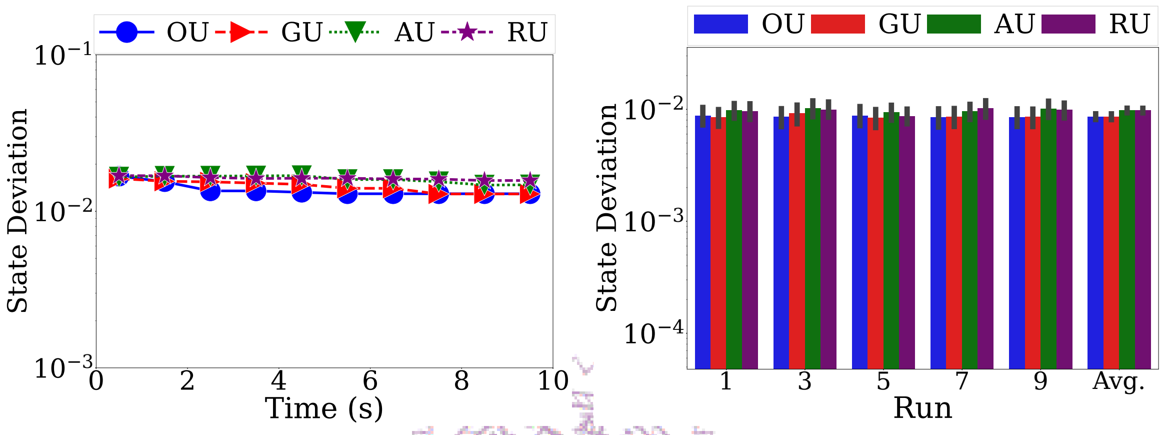
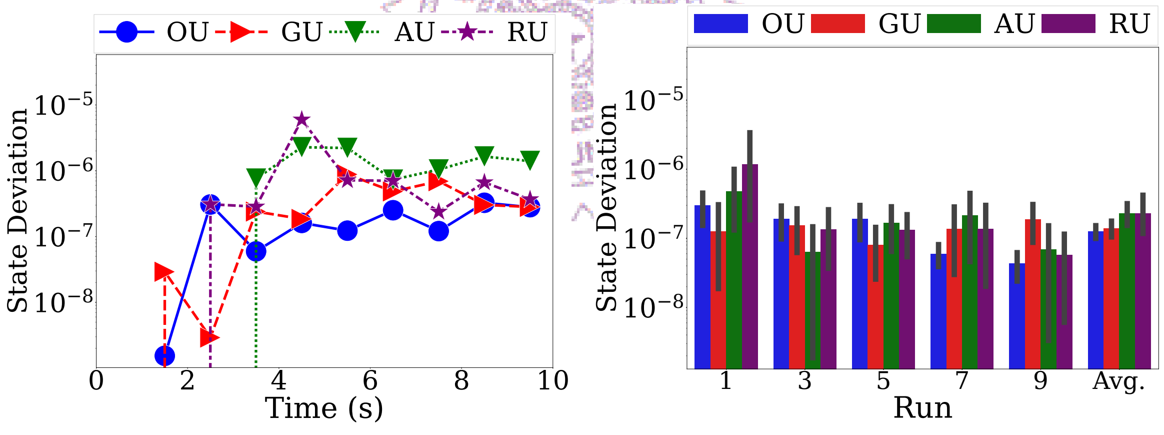


Figure 7.8: State deviation from different state synchronization algorithms under 1 Mbps networked device bandwidth. Expected state deviation: (a) from a network switch in a sample run and (b) across all networked devices. Actual state deviation: (c) from a network switch in a sample run and (d) across all networked devices.



(a)

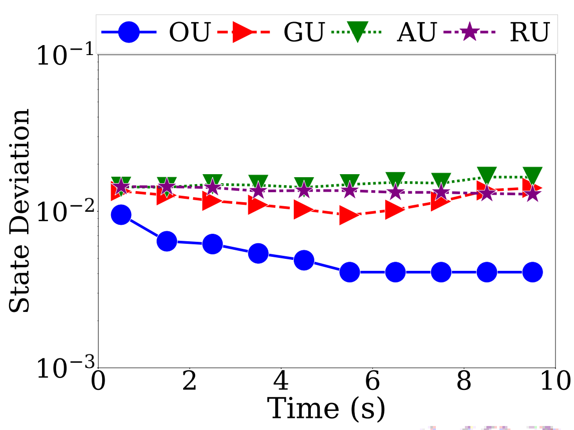
(b)



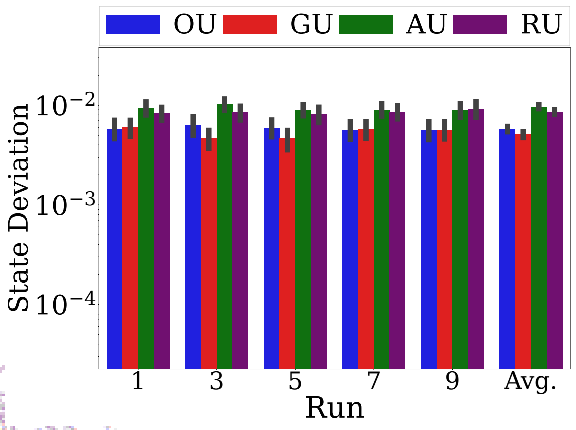
(c)

(d)

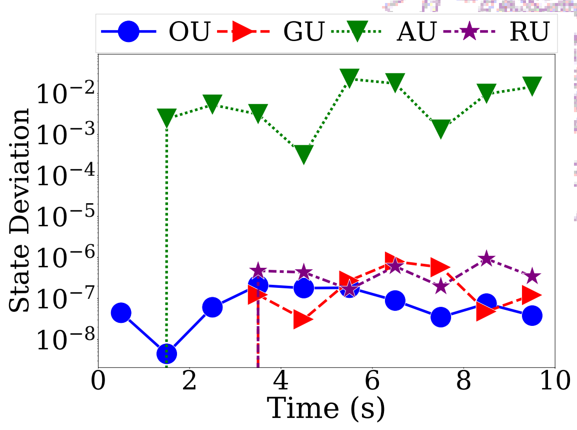
Figure 7.9: State deviation from different state synchronization algorithms under 0.5 Mbps networked device bandwidth. Expected state deviation: (a) from a network switch in a sample run and (b) across all networked devices. Actual state deviation: (c) from a network switch in a sample run and (d) across all networked devices.



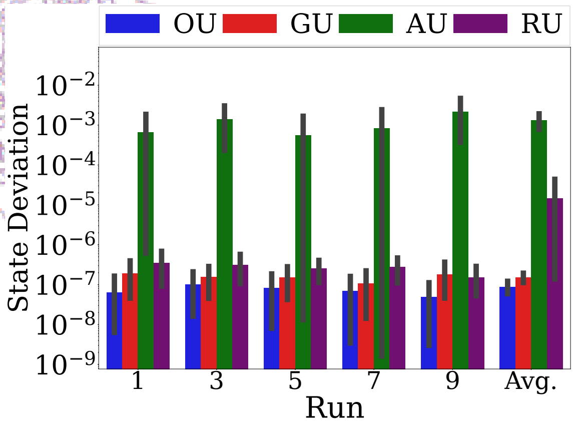
(a)



(b)

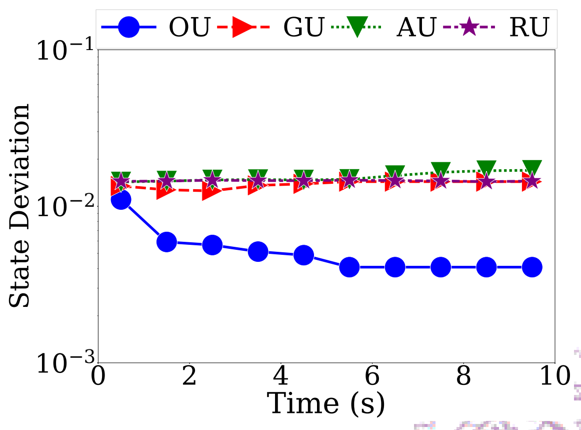


(c)

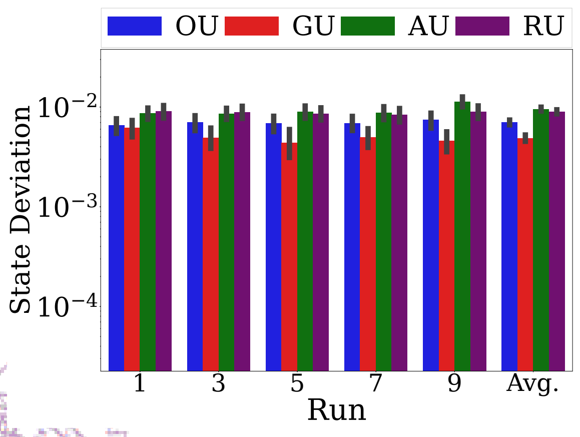


(d)

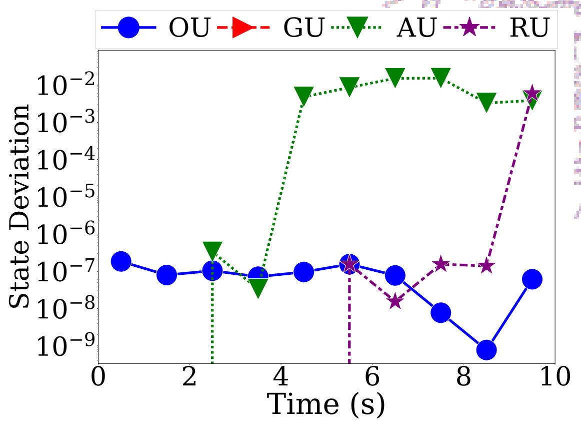
Figure 7.10: State deviation from different state synchronization algorithms under 15 Mbps control server bandwidth. Expected state deviation: (a) from a network switch in a sample run and (b) across all networked devices. Actual state deviation: (c) from a network switch in a sample run and (d) across all networked devices.



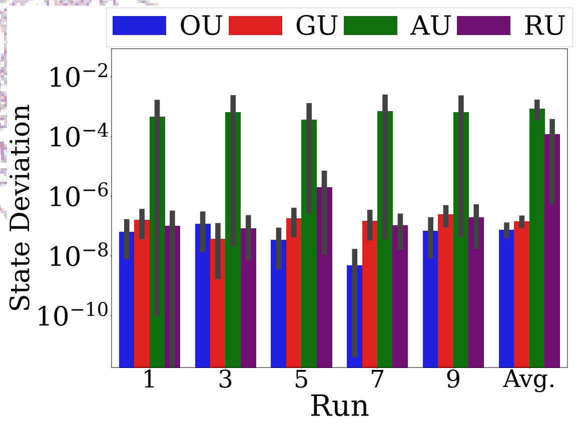
(a)



(b)

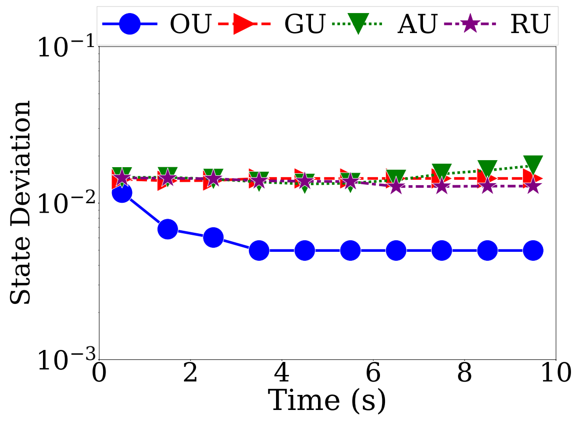


(c)

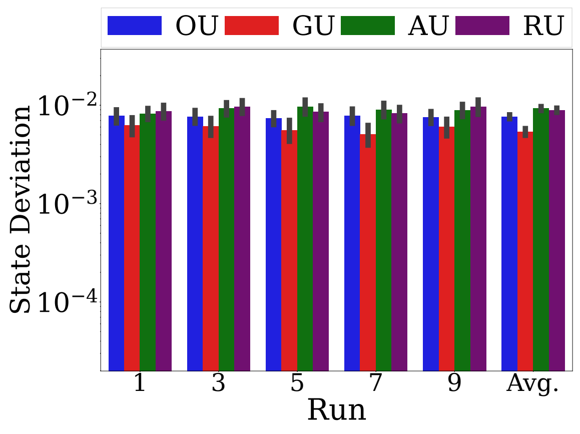


(d)

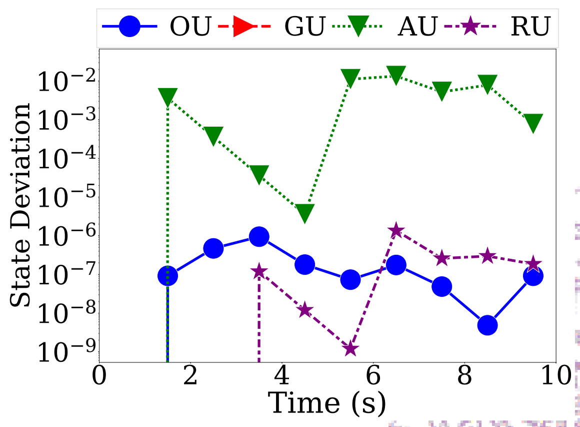
Figure 7.11: State deviation from different state synchronization algorithms under 10 Mbps control server bandwidth. Expected state deviation: (a) from a network switch in a sample run and (b) across all networked devices. Actual state deviation: (c) from a network switch in a sample run and (d) across all networked devices.



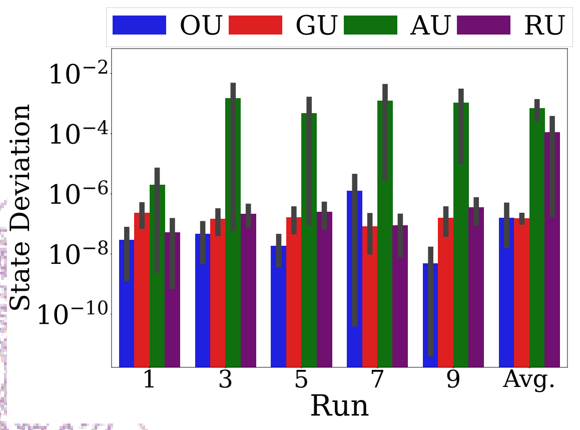
(a)



(b)

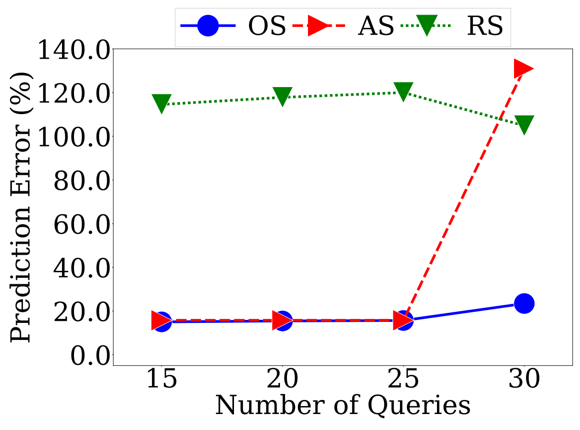


(c)

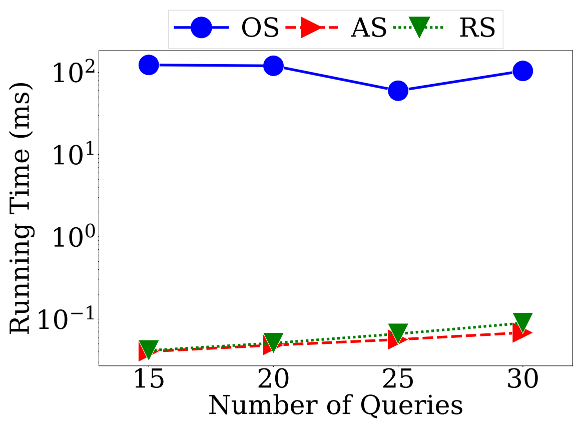


(d)

Figure 7.12: State deviation from different state synchronization algorithms under 5 Mbps control server bandwidth. Expected state deviation: (a) from a network switch in a sample run and (b) across all networked devices. Actual state deviation: (c) from a network switch in a sample run and (d) across all networked devices.

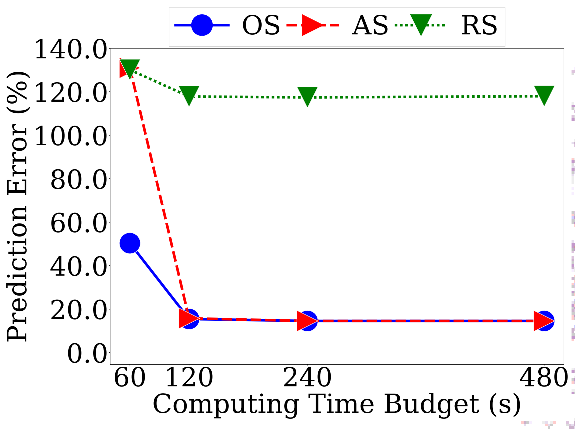


(a)

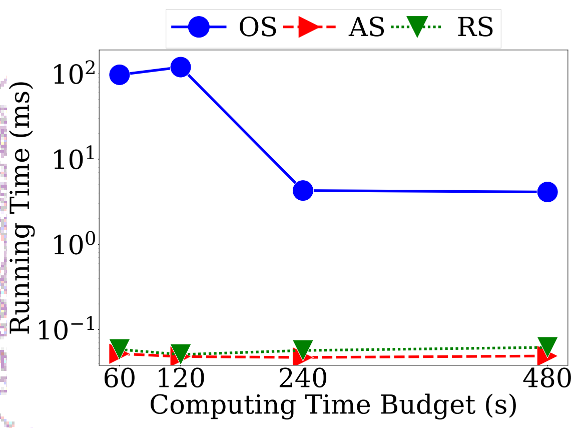


(b)

Figure 7.13: Performance of different what-if analysis algorithms under different numbers of queries: (a) prediction error and (b) running time.



(a)



(b)

Figure 7.14: Performance of different what-if analysis algorithms under different computing time budgets: (a) prediction error and (b) running time.

Chapter 8

Conclusion

In this chapter, we summarize the key contributions of our work, highlighting the successful development and optimization of an NDTC for IoT-instrumented smart environments. We also discuss potential future research directions to enhance the scalability and functionality of our proposed solutions.

8.1 Concluding Remarks

We developed and optimized an NDTC for IoT-instrumented smart environments, which has not been thoroughly studied in the literature. By extending an open-source SDN controller, we constructed a DT-enabled smart environment that efficiently supports multiple innovative applications. Our proposed OU and GU state synchronization algorithms effectively minimize state deviation between PTs and their DTs by optimizing the update frequency and data granularity within the network bandwidth budgets. Our OS what-if analysis algorithm accurately selects suitable what-if analyzers for individual queries, minimizing QoS prediction errors under the computing time budget. Our extensive experiments in a real smart environment testbed reveal the merits of our proposed algorithms over the baseline ones. For example, our OU and GU algorithms outperform the baseline algorithms in terms of state deviation: 99.49% and 94.92% reductions are observed. In addition, our OS algorithm outperforms the AS and RS algorithms in terms of prediction error: more than 75.77% and 83.63% reductions are observed.

8.2 Future Work

Our work in this thesis can be extended in the following directions:

- *Network topology with larger scale.* In our work, we develop and evaluate our NDTC and state synchronization algorithms on a testbed containing seven networked devices. In the future, we plan to implement our system in larger network topologies with more networked devices to evaluate the scalability of the NDTC and our state synchronization algorithm.

- *Human-in-the-loop suggestion from what-if analyzers.* In our work, we focus on the implementation of four what-if analyzers. These analyzers utilize the network topology and the states of DTs from our NDTC to conduct analysis and predict QoS metrics. In the future, we plan to incorporate human-in-the-loop suggestions for these analyzers, providing administrators with recommendations for new deployment plans or settings. These suggestions can assist in enhancing the performance of smart environments.

- *Domain-specific simulators.* In our work, we use what-if analyzers to predict QoS metrics and evaluate the network performance of smart environments. Additionally, different types of what-if analyzers can be implemented with DTs. For example, SUMO is a traffic simulator that can model urban traffic flow, while QBlade is a wind turbine simulator that can simulate design and electricity generation under various settings. These what-if analyzers can leverage DTs in smart environments to provide analysis that help administrators evaluate different configurations and settings.

Bibliography

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4):2347–2376, June 2015.

[2] K. Alanezi and S. Mishra. Towards a Scalable Architecture for Building Digital Twins at the Edge. In *Proc. of IEEE/ACM Symposium on Edge Computing (SEC)*, pages 325–329, Wilmington, DE, USA, December 2023.

[3] P. Almasan, M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, D. Perino, D. López, A. A. P. Perales, P. Harvey, L. Ciavaglia, L. Wong, et al. Network digital twin: Context, enabling technologies, and opportunities. *IEEE Communications Magazine*, 60(11):22–27, 2022.

[4] A. Benczúr, L. Kocsis, and R. Pálovics. Online machine learning in big data streams. *arXiv preprint arXiv:1802.05872*, pages 1–40, 2018.

[5] H. Chen, T. Todd, D. Zhao, and G. Karakostas. Digital Twin Model Selection for Feature Accuracy. *IEEE Internet of Things Journal*, 11(7):11415 – 11426, 2023.

[6] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang. Deep reinforcement learning for stochastic computation offloading in digital twin networks. *IEEE Transactions on Industrial Informatics*, 17(7):4968–4977, 2020.

[7] Y. Edalat, J.-S. Ahn, and K. Obraczka. Smart experts for network state estimation. *IEEE Transactions on Network and Service Management*, 13(3):622–635, 2016.

[8] A. El Saddik, F. Laamarti, and M. Alja'Afreh. The potential of digital twins. *IEEE Instrumentation and Measurement Magazine*, 24(3):36–41, May 2021.

[9] M. Ferriol-Galmés, J. Suárez-Varela, J. Paillissé, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio. Building a digital twin for network optimization using graph neural networks. *Elsevier Computer Networks*, 217:109329, 2022.

[10] Free Software Foundation. GLPK, 2012. `https://www.gnu.org/software/glpk/`.

[11] A. Fuller, Z. Fan, C. Day, and C. Barlow. Digital twin: Enabling technologies, challenges and open research. *IEEE access*, 8:108952–108971, May 2020.

[12] C. Güemes-Palau, P. Almasan, S. Xiao, X. Cheng, X. Shi, P. Barlet-Ros, and A. Cabellos-Aparicio. Accelerating deep reinforcement learning for digital twin network optimization with evolutionary strategies. In *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–5, Budapest, Hungary, April 2022.

[13] C. Hoi, D. Sahoo, J. Lu, and P. Zhao. Online learning: A comprehensive survey. *Elsevier Neurocomputing*, 459:249–289, 2021.

[14] H. Hong, Q. Wu, F. Dong, W. Song, R. Sun, T. Han, C. Zhou, and H. Yang. Netgraph: An intelligent operated digital twin platform for data center networks. In *Proc. of the ACM SIGCOMM workshop on network-application integration*, pages 26–32, Virtual Event, USA, August 2021.

[15] International Business Machines Corporation (IBM). CPLEX, 2024. `https://www.ibm.com/products/ilog-cplex-optimization-studio`.

[16] ITU-T. Digital twin network – Requirements and architecture, 2022. `https://www.itu.int/rec/T-REC-Y.3090s`.

[17] M. Jafari, A. Kavousi-Fard, T. Chen, and M. Karimi. A review on digital twin technology in smart grid, transportation system and smart city: Challenges and future. *IEEE Access*, 11:17471 – 17484, February 2023.

[18] e. a. Jin, Jiong. Network architecture and QoS issues in the Internet of things for a smart city. In *Proc. of IEEE International Symposium on Communications and Information Technologies (ISCIT)*, pages 956–961, October 2012.

[19] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, December 2014.

[20] G. Li, T. Luan, X. Li, J. Zheng, C. Lai, Z. Su, and K. Zhang. Breaking down data sharing barrier of smart city: A digital twin approach. *IEEE Network*, pages 1 – 9, May 2023.

[21] J. Li, J. Wang, Q. Chen, Y. Li, and A. Zomaya. Digital twin-enabled service satisfaction enhancement in edge computing. In *Proc. of IEEE INFOCOM Conference on Computer Communications*, pages 1–10, New York City, NY, USA, May 2023.

[22] G. Lin, J. Gel, Y. Wu, H. Li, and L. Li. Digital twin networks: learning dynamic network behaviors from network flows. In *Proc. of IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, Rhodes, Greece, June 2022.

[23] Linux Foundation. OpenvSwitch, 2016. `https://www.openvswitch.org/`.

[24] Q. Liu, L. Tang, T. Wu, and Q. Chen. Deep reinforcement learning for resource demand prediction and virtual function network migration in digital twin network. *IEEE Internet of Things Journal*, 10(21):19102 – 19116, 2023.

[25] P. Major, G. Li, H. Hildre, and H. Zhang. The use of a data-driven digital twin of a smart city: A case study of Ålesund, norway. *IEEE Instrumentation & Measurement Magazine*, 24(7):39–49, 2021.

[26] O. Marai, T. Taleband, and J. Song. Roads infrastructure digital twin: A step toward smarter cities realization. *IEEE Network*, 35(2):136–143, March-April 2021.

[27] K. Mayer, R. Pinto, J. Soares, D. Arantes, C. Rothenberg, V. Cavalcante, L. Santos, F. Moraes, and D. Mello. Demonstration of ML-assisted soft-failure localization based on network digital twins. *IEEE Journal of Lightwave Technology*, 40(14):4514–4520, 2022.

[28] R. Minerva and N. Crespi. Digital twins: Properties, software frameworks, and application scenarios. *IT Professional*, 23(1):51–55, April 2021.

[29] G. Mylonas, A. Kalogeras, G. Kalogeras, C. Anagnostopoulos, C. Alexakos, and L. Muñoz. Digital twins from smart manufacturing to smart cities: A survey. *IEEE Access*, 9:143222–143249, October 2021.

[30] Y. Nabil, H. ElSawy, S. Al-Dharrab, H. Mostafa, and H. Attia. Data aggregation in regular large-scale IoT networks: Granularity, reliability, and delay tradeoffs. *IEEE Internet of Things Journal*, 9(18):17767–17784, 2022.

[31] H. Nguyen, R. Trestian, D. To, and M. Tatipamula. Digital twin for 5G and beyond. *IEEE Communications Magazine*, 59(2):10–15, February 2021.

[32] NS3 Network Simulator , 2011. `https://www.nsnam.org/`.

[33] Ontje Lünsdorf. Simpy, 2023. `https://simpy.readthedocs.io/en/latest/`.

[34] M. Polverini, I. Germini, A. Cianfrani, F. G. Lavacca, and M. Listanti. A Digital Twin based Framework to Enable "What-If" Analysis in BGP Optimization. In *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–6, Miami, FL, USA, May 2023.

[35] M. Polverini, F. Lavacca, J. Galán-Jiménez, D. Aureli, A. Cianfrani, and M. Listanti. Digital twin manager: A novel framework to handle conflicting network applications. In *Proc. of IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 85–88, Phoenix, AZ, USA, November 2022.

[36] K. Poularakis, Q. Qin, L. Ma, S. Kompella, K. K. Leung, and L. Tassiulas. Learning the optimal synchronization rates in distributed SDN control architectures. In *Proc. of IEEE INFOCOM 2019 Conference on Computer Communications*, pages 1099–1107, Paris, France, April 2019.

[37] L. Raes, P. Michiels, T. Adolphi, C. Tampere, A. Dalianis, S. McAleer, and P. Kogut. Duet: A framework for building interoperable and trusted digital twins of smart cities. *IEEE Internet Computing*, 26(3):43–50, May 2022.

[38] D. Raj, T. Ahmed, A. Hirwe, P. Tammana, and K. Kataoka. Building a digital twin network of sdn using knowledge graphs. *IEEE Access*, 11:63092 – 63106, 2023.

[39] A. Rasheed, O. San, and T. Kvamsdal. Digital twin: Values, challenges and enablers from a modeling perspective. *IEEE Access*, 8:21980–22012, January 2020.

[40] A. Ridwan, M. Radzi, F. Abdullah, and Y. Jalil. Applications of machine learning in networking: a survey of current issues and future challenges. *IEEE access*, 9:52523–52556, 2021.

[41] M. M. Roselló, V. Cancela, I. Quintana, and M. Lorenzo. Network Digital Twin for Non-Public Networks. In *Proc. of IEEE 24th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 495–500, Boston, MA, USA, June 2023.

[42] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio. Unveiling the potential of graph neural networks for network modeling and optimization in sdn. In *Proceedings of the 2019 ACM Symposium on SDN Research*, pages 140–151, 2019.

[43] Ryu SDN Framework Community. Ryu, 2017. `https://ryu-sdn.org/`.

[44] N. Savage. Virtual duplicates. *Commun. ACM*, 65(2):14–16, January 2022.

[45] K. Sood, K. K. Karmakar, S. Yu, V. Varadharajan, S. R. Pokhrel, and Y. Xiang. Alleviating heterogeneity in SDN-IoT networks to maintain QoS and enhance security. *IEEE Internet of Things Journal*, 7(7):5964–5975, December 2019.

[46] M. Talebkhah, A. Sali, M. Marjani, M. Gordan, S. J. Hashim, and F. Z. Rokhani. Iot and big data applications in smart cities: recent advances, challenges, and critical issues. *IEEE Access*, 9:55465–55484, April 2021.

[47] F. Tang, X. Chen, K. Rodrigues, M. Zhao, and N. Kato. Survey on digital twin edge networks (DITEN) toward 6G. *IEEE Open Journal of the Communications Society*, 3:1360–1381, August 2022.

[48] G. Tangari, D. Tuncer, M. Charalambides, Y. Qi, and G. Pavlou. Self-adaptive decentralized monitoring in software-defined networks. *IEEE Transactions on Network and Service Management*, 15(4):1277–1291, 2018.

[49] S. Taylor, B. Johansson, S. Jeon, L. Lee, P. Lendermann, and G. Shao. Using simulation and digital twins to innovate: Are we getting smarter? In *2021 Winter Simulation Conference (WSC)*, pages 1–13, December 2021.

[50] M. Vaezi, K. Noroozi, T. Todd, D. Zhao, G. Karakostas, H. Wu, and X. Shen. Digital twins from a networking perspective. *IEEE Internet of Things Journal*, 9(23):23525–23544, 2022.

[51] R. Vilalta, L. Gifre, R. Casellas, R. Muñoz, R. Martínez, A. Mozo, A. Pastor, D. López, and J. P. Fernández-Palacios. Applying digital twins to optical networks with cloud-native sdn controllers. *IEEE Communications Magazine*, 61(12):128 – 134, 2023.

[52] H. Wang, Y. Wu, G. Min, and W. Miao. A graph neural network-based digital twin for network slicing management. *IEEE Transactions on Industrial Informatics*, 18(2):1367–1376, 2020.

[53] Y. Wang, Z. Su, S. Guo, M. Dai, T. H. Luan, and Y. Liu. A survey on digital twins: Architecture, enabling technologies, security and privacy, and future prospects. *IEEE Internet of Things Journal*, 10(17):14965–14987, 2023.

[54] J. Wen, B. Gabrys, and K. Musial. Toward digital twin oriented modeling of complex networked systems and their dynamics: A comprehensive survey. *Ieee Access*, 10:66886–66923, 2022.

[55] J. Wieme, M. Baert, and J. Hoebeke. Managing a QoS-enabled Bluetooth Mesh network using a Digital Twin Network: An experimental evaluation. *Elsevier Internet of Things*, 25:101023, 2024.

[56] Y. Wu, K. Zhang, and Y. Zhang. Digital twin networks: A survey. *IEEE Internet of Things Journal*, 8(18):13789–13804, 2021.

[57] M. Yannuzzi, F. van Lingen, A. Jain, O. L. Parellada, M. M. Flores, D. Carrera, J. L. Pérez, D. Montero, P. Chacin, A. Corsaro, et al. A new era for cities with fog computing. *IEEE Internet Computing*, 21(2):54–67, 2017.

[58] P. Yu, J. Zhang, H. Fang, W. Li, L. Feng, F. Zhou, P. Xiao, and S. Guo. Digital twin driven service self-healing with graph neural networks in 6g edge networks. *IEEE Journal on Selected Areas in Communications*, 41(11):3607 – 3623, 2023.

[59] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, August 2014.

[60] L. Zhu, M. M. Karim, K. Sharif, C. Xu, F. Li, X. Du, and M. Guizani. SDN controllers: A comprehensive analysis and performance evaluation study. *ACM Computing Surveys (CSUR)*, 53(6):1–40, December 2020.