Digital Twin Placement for Minimum Application Request Delay with Data Age Targets

Digital Twin Placement for Minimum Application Request Delay with Data Age Targets

By Mehrad Vaezi

A Thesis Submitted to the School of Graduate Studies in the Partial Fulfillment of the Requirements for the Master of Applied Sciences Degree.

McMaster University

Master of Applied Sciences (2022)

Hamilton, Ontario (Department of Electrical and Computer Engineering)

TITLE: Digital Twin Placement for Minimum Application Request Delay with Data Age Targets

AUTHOR: Mehrad Vaezi (McMaster University)

SUPERVISOR: Dr. Dongmei Zhao

CO-SUPERVISORS: Dr. George Karakostas and Dr. Terence D. Todd

NUMBER OF PAGES: x, 47

Abstract

Digital Twins are softwarized mirrors of physical systems. They can represent their corresponding physical counterparts in real-world applications and reflect the behavior of the latter under different scenarios with decent accuracy. In this thesis, we consider the case where an application requests data from multiple digital twins, each representing a physical system. The digital twins are hosted on execution servers located between the application and the set of physical devices. Each digital twin has to be periodically updated by its physical system and uses a portion of the execution server's computing resource to refresh itself. Due to the scarcity of computation resources of the execution servers, in this thesis, we have tackled the problem of optimal digital twin placement onto a limited set of execution servers. We are aiming at minimizing the latency of the digital twins' responses to the application's requests while keeping the age of information of served data below a certain threshold. We first formulate the problem as an integer quadratic program (IQP) and then transform it into a semidefinite program (SDP). We prove that the problem is NP-complete and propose polynomial-time approximation algorithms that solve the problem with different trade-offs between the accommodation of the application's request latency and the achievement of data age targets.

Acknowledgements

I would like to express my gratitude to my supervisors, Professors Dongmei Zhao, George Karakostas, and Terence D. Todd who guided me throughout my graduate studies. I am also grateful to Dr. Douglas Down for his valuable comments as part of the examining committee. Finally, I would also like to thank my friends and family for their love and support.

Contents

List of Figures

List of Symbols

$\mathcal{M}$	Set of physical systems
$\mathcal{N}$	Set of execution servers
$d_{m,n}^{data}$	Transmission delay of updates from PS m to ES n
$d_{m,n}^{down}$	Transmission delay of application server's request for data from DT m at ES n
$d_{m,n}^{up}$	Transmission delay of the response for application server's request from DT m at ES n
$c_{m,n}$	Processing time of the update of DT m at ES n
T_m	Update period of DT m
$A^\star$	Application's maximum age-of-information tolerance
X_{mn}	Decision variable for placing DT m on ES n

List of Abbreviations

DT	Digital Twin
PS	Physical System
ES	Execution Server
AS	Application Server
AoI	Age-of-Information

Declaration of Authorship

I, Mehrad Vaezi, declare that this thesis titled, "Digital Twin Placement for Minimum Application Request Delay with Data Age Targets" and the work presented in it are my own.

Chapter 1

Introduction

1.1 Digital Twins and Definitions

The Digital Twin (DT) of a Physical System (PS) is a softwarized representation of that system. The concept was initially introduced by Michael Grieves for lifecycle management of products [1], [2] in manufacturing environments. Digital twins were built to support simulation and prediction of the behavior of the physical systems from their design and manufacturing to production and deployment until their end of life and disposal.

The PS-DT relationship consists of three main elements:

- Physical System
- Digital Twin (Virtual System)
- Linkage

The physical system is broken into measurable features/attributes. For instance, an aircraft has countless features such as speed, acceleration, altitude, engine temperature, cabin air pressure etc., or a moisture meter that contains the average humidity of the

environment. The digital counterpart of the physical system is updated with the current values of the features through the linkage medium and would reflect the same set of features and attributes of the PS. The linkage is the synchronization bridge between the PS and its DT. It is either one-directional in cases where the PS updates the DT but receives no feedback from it [1] or bi-directional where the DT provides feedback to the PS itself. Based on the PS-DT distance and time sensitivity of the application, the linkage may be wired (e.g. a production line connected to a central computer in a manufacturing environment) or wireless (e.g. an aircraft updating its DT in a cloud server through satellite communication).

Ideally, a digital twin has to mirror every aspect of the physical system in real time. That is, if the PS and its DT are triggered with the same input, they should produce completely identical outputs at the same time. This, however, is often not feasible for the DT of a complex physical system that consists of a large number of fast-changing features. In practical scenarios, the quality of a DT can be affected by the available resources including:

- Storage capacity
- Computation resources
- Network resources and communication bandwidth

As a case scenario, a moving vehicle contains numerous features such as its current location, destination, velocity, available fuel, water temperature, tire air pressure, etc. A functional DT of that vehicle has to store the past and present values of those features in a large database. Furthermore, the DT would have to undergo substantial processing of that data to come up with accurate predictions for the future status of the vehicle.

Typically, a DT can be located miles away from the PS inside a cloud server. In this

[1] In this case the DT responds to the requests of a third-party application representing the PS.

case, a certain amount of network bandwidth has to be allocated to the transmission of updates from the PS to its DT. Time-sensitive applications require higher update frequencies from the PS and have stringent latency constraints. These requirements result in larger bandwidth consumption and require efficient network resource management algorithms.

Consequently, creating a software representation that reflects every feature of the physical system in real time requires massive amounts of communication, computation, and storage resources. For that reason, based on the specifications of the application, a subset of the features of a PS is reflected by its digital twin with a trade-off between resource consumption and the performance of the DT.

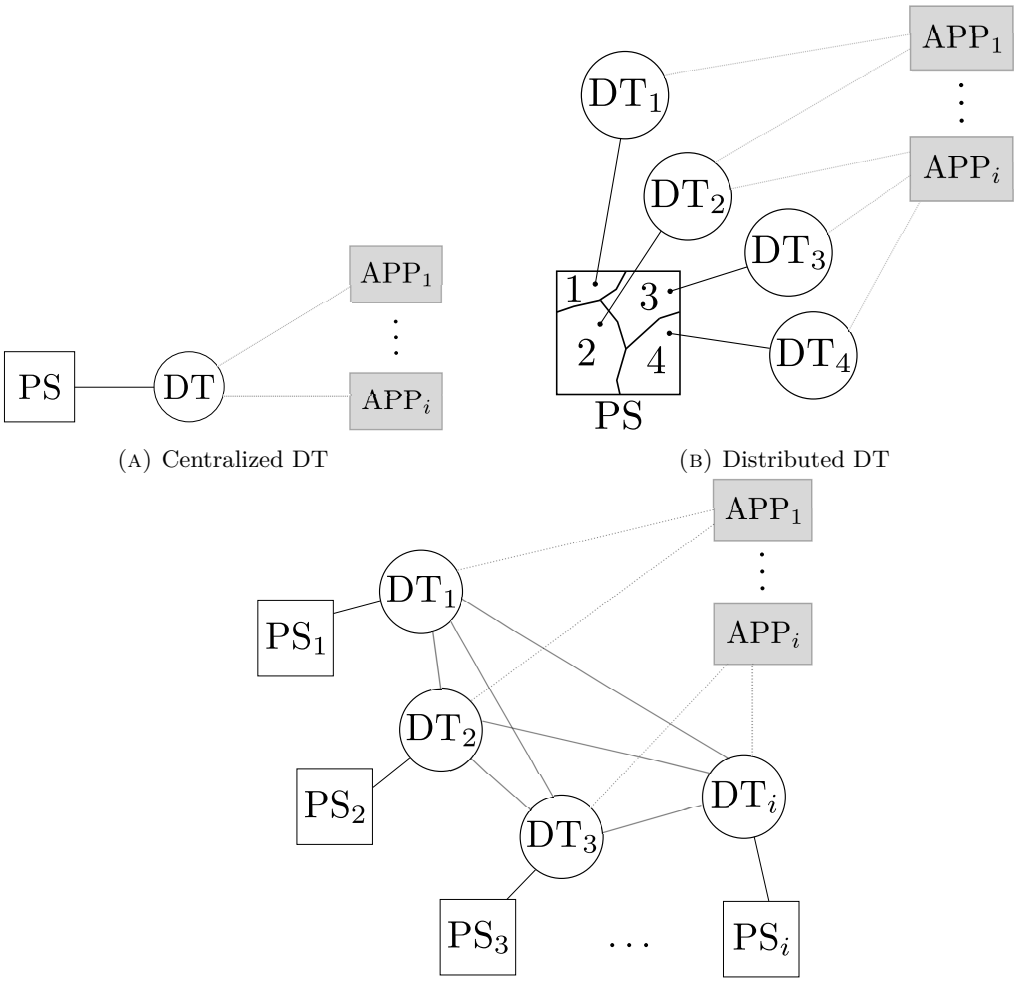
1.2 Digital Twin Placement Architectures

In real-world use cases of digital twins, the DTs are often hosted inside servers that are located between the physical system and the application servers. The digital twin acts as an intermediary representation of the physical system and responds to the requests of the applications using the data obtained from the PS. Depending on the requirements of the applications and the complexity of the physical device, multiple digital twin placement architectures can be visualized.

Centralized DT is the most basic architecture where only one server hosts the DT of the physical system. Depending on the application's demands, the DT may partially mirror the PS or reflect the entire features in the physical device's feature set. The choice of the DT's hosting server can be made based on the latency and available computation capacity of the candidate servers. Figure 1(a) depicts this placement architecture.

For complex physical systems, there can be multiple digital twins each reflecting a segment of the PS. A *Distributed DT* architecture can be applied in cases when different

applications are interested in different subsets of the features of the PS. For instance, a traffic management application might need information regarding the speed and location of the vehicles while a predictive maintenance application is interested in real-time data from internal parts of the vehicles. Nevertheless, this architecture can also be used when the DTs are scattered across several edge servers for the sake of distributing the computation load of the digital twins. Figure 1(b) depicts this placement architecture.



(A) Centralized DT

(B) Distributed DT

(C) Network of DTs

Figure 1: DT Placement Architectures.

With the advent of the IoT technologies, end devices are interacting with each other more frequently. In a *Network of DTs* architecture, digital twins of end devices communicate with other DTs on behalf of the physical devices. This gives the end devices the opportunity of using the computation resources of hosting servers and speed up their communication through high-speed backhaul links of those servers. An example for this architecture is collaborative driving, where the digital twin of a vehicle collaborates with the DTs of other cars and obtains an understanding of the whole environment [3]. This architecture has been also used in space-air-ground networks (SAGIN) [4], [5].

A *Network DT* (digital twin of a network) is a single entity that represents an entire network. A network DT can be built by aggregating the information of a network of DTs which can later be used for maintenance of the network in general. Network DTs are especially applicable in the trending topic of software-defined networking (SDN) where the network DT is accessed by the central network controller in order to gather information for network management decisions and maintenance.

1.3 Thesis Contributions and Organization

In real-world scenarios, the digital twins can be placed between an application and the physical device and communicate with the application on behalf of the PS. The application server would request fresh status information of the physical device, e.g., real-time location and speed of a vehicle. Using the digital twins, the PS would periodically update its digital twin while the application server receives responses from the DT with lower latency. This resembles the centralized architecture of Figure 1(a) discussed in the previous section.

Time-sensitive applications impose constraints on the Age-of-Information (AoI) of the data that they receive from the digital twins. In other words, data that is not fresh enough will be rendered useless by the application. As the number of the physical

devices increase, the scarcity of available hosting servers that satisfy the AoI target of the application becomes evident and a crucial issue. Therefore, the candidate servers have to be efficiently allocated to the digital twins such that the physical devices can update their DTs on time while the DT sends fresh enough data to the application server according to its requirements.

There are prior works such as [6], [7] where digital twins are placed on a set of servers to serve particular applications. Wang *et al.* [6] study a similar system model for optimal placement of service entities in a virtual reality (VR) application. VR service entities differ with digital twins in the sense that DTs have to be periodically updated by their physical systems. Since the received data needs to be processed before the next update arrives, this adds an update constraint for the digital twins which does not exist for the VR service entities. Their optimization problem aims at minimizing the service delay of the entities modeled as the placement cost of the hosting edge servers. They, however, do not take into account the AoI tolerance constraint of the application. Furthermore, they use a naive linear constraint to model the CPU sharing among the service entities resulting in an integer linear program (ILP), while in this work, the DTs share the server's CPU according to a time-sharing policy which will make the problem an integer quadratic program (IQP).

Lu *et al.* [7] have presented deep-learning-based algorithms for a similar digital twin placement problem. A set of digital twin server locations is first obtained and the digital twins are then migrated between these servers when needed. They assume that digital twins are light, and, therefore, homogeneous in the sense that they are all of the same size, which may not hold in general. The computational resource allocation model also does not account for the time each digital twin consumes the server CPU. The optimization problem for placement uses the average user-DT interaction delay as its objective which can result in unfair resource allocation policies.

In this thesis, we address the aforementioned problem of efficient digital twin placement. Specifically, the system model consists of an application, the set of physical devices, and the set of host servers located between them. Our objective is to minimize the response time of the host servers to the application. Furthermore, the DT placement policy has to accommodate the AoI tolerance of the application server. We employ a time-sharing resource allocation model for computational resources at the servers which results in an integer quadratic problem. The integer quadratic program (IQP) is transformed into a semidefinite program (SDP). It is subsequently proven that the problem is NP-complete. As a result, we have proposed polynomial-time approximation algorithms that solve the problem with trade-offs between the adaptation of the application's data age target and the data serving time from the hosting servers to the application. The performance of the proposed approximation algorithms are compared with the globally optimum solution through computer simulations.

The rest of the thesis is organized as follows. Chapter 2 reviews the available literature on digital twins and applications that have exploited this concept. Chapter 3 introduces the system model and formulates an optimization problem that describes the objective and constraints of the aforementioned problem. Chapter 4 describes our solution approach and proposes approximation algorithms that are used for obtaining a solution for the problem. Chapter 5 presents simulation results and their analysis and evaluation. Chapter 6 discusses possible future work and concludes the thesis.

Chapter 2

Literature Review

In this chapter, we review the literature related to digital twins. Papers on digital twins can be divided into two main categories, namely, papers that propose a method to design and implement a digital twin for a certain physical system such as a factory or an airplane, and those that use the digital twins in their general terms and definitions to optimize and enhance a series of operations such as routing and mobile computation offloading. In the first section of this chapter, we review the work done on smart manufacturing and industry. The second section is dedicated to reviewing the papers that employ digital twins in machine learning applications. The final section reviews the state of the art on optimizing network performance using digital twins.

2.1 Digital Twins in Industry

The concept of digital twins was first introduced for industrial production by Michael Grieves in [1], [2]. The objective was to establish a technology that could imitate the entire lifecycle of a product from the initial stages of design and testing to the production and operation and ultimately disposal of the product. Since then, digital twins have become increasingly popular in industry 4.0 and smart manufacturing. Digital twins

combined with big data analytics provide a test bed for simulation and fault diagnosis of products before the actual manufacturing takes place. [2], [8]–[12].

Cyber-physical systems are the main trend in industry 4.0 and smart manufacturing. These are systems that consist of a physical system and a virtual system. The two systems are in constant interaction with each other. Any major change that will be applied to the physical system, needs to be tested and analyzed beforehand in the virtual system. Therefore, both systems have to evolve together over time and remain synchronized [13], [14]. Since modern cyber-physical systems require scalable storage, computation, and communication capabilities, they are often deployed on cloud-based architectures. The key properties of a cloud-based cyber-physical system, namely, computation, control, and communication are analytically described in [15]. They propose a reference model for cloud-based cyber-physical systems that can be employed in the design of such systems.

Starting with the paper published by NASA [16], aviation applications also began to use DTs. In [16], [17] it is mentioned that prior to using DTs, aircraft sustainment management was done based on statistical distributions of material properties, heuristic design philosophies, and physical testing of the aircraft. Due to the inefficiency of those methods under extreme requirements and dynamic environments, DTs attracted more popularity in aircraft design. Digital twin applications in aviation include ultra-high fidelity simulations of structural deflections of the aircraft in response to dynamic flight conditions which enables mirroring the entire life-cycle of the physical object.

2.2 Digital Twins and Machine Learning

Machine learning is a popular technique in the digital twin paradigm. A DT can host and train a learning model and use it in its decision-making and predictions. Reinforcement learning in particular is useful when the PS provides feedback on the predictions made

by the DT. In [18], the DT of a network acts as a critic and provides feedback to the decisions made by a reinforcement learning model. The model is subsequently used for making optimal offloading decisions for the IoT devices in the network.

In [19], digital twins of manufacturing cells are created and used for simulating system behavior and predicting possible faults. The twin sends automation commands to the physical system consisting of sensors, PLC, and actuators and receives feedback signals in return. In its simulation, the DT utilizes a deep reinforcement learning model which uses the feedback signals to come up with efficient manufacturing strategies.

Federated learning, as a modern approach in machine learning, preserves the privacy of users and their information. This is attained by allowing the users to train a learning model on their own devices and send only the model to a centralized server instead of feeding their raw data to the server responsible for training the model. This can furthermore reserve a great amount of communication and computation resources for the network. In [20], the digital twin of an edge network responsible for user scheduling and bandwidth allocation updates its state through the learning models sent by the end devices. The devices train a federated learning model of their own and eventually these models are aggregated inside the digital twin server.

In [21], a digital twin reflecting the dynamic characteristics of an air-ground network is designed. Ground devices, including vehicles and mobile phones, train their local federated learning models. These local learning models are aggregated in the air network consisting of several drones. The air network, containing computation resources that can be used for offloading the ground network's computational tasks, updates its digital twin, which subsequently decides on offloading policies of the ground devices.

In [22], the digital twin models the slices of the shared resources of a network using a graph neural network. The inter-dependencies of the slices are captured by the graph

neural network which helps the digital twin to predict the generated traffic of each slice and estimate the end-to-end metrics of the slices.

2.3 Network Performance Enhancement using Digital Twins

Modern networks consist of a vast number of devices and intermediate nodes. Due to the heterogeneity and dynamic nature of these networks together with the time-sensitivity of the applications that use those networks, efficient network management algorithms, and intelligent technologies have been made use of in the control of these networks. In the last five years, digital twins have become increasingly popular in the network management literature by providing real-time information and perspective of the network to the network controller [23].

The major part of the applications of DTs in network management has been on edge computing and task offloading of mobile devices. Digital twins of mobile users are created and stored and periodically updated in a nearby edge server. A central network controller learns about the current state of the network and resource demands of mobile devices by contacting the edge servers that are hosting the digital twins. The central controller can then decide on the offloading policies and computational resource allocation schemes based on the learned information [24], [25].

Vehicular networks have also benefited from digital twins. Internet of Vehicles (IoV) consists of a large number of connected vehicles that either distributively share their real-time states or update a central server with their states so that each vehicle can obtain a picture of its surrounding environment and use it in its decision makings. Reference [26] has proposed an architecture for the network of connected vehicles that is built upon the virtual representation of the vehicles (DTs). This architecture is used by traffic administrators for traffic scheduling and prediction. A DT-based architecture for a network of autonomous driving vehicles is proposed in [3] and [27]. Every autonomous

vehicle is represented by its digital twin in the virtual space where DTs collaborate to proactively decide on the driving policies of the vehicles.

Reference [28] combines deep learning with digital twins for a vehicular edge computing management framework where cars can offload their computational tasks to vehicles with computing power. The digital twin of the network holds the digital twins of cars reflecting the cooperation between them. It is equipped with a multi-agent deep learning engine responsible for making offloading decisions while taking the inter-vehicle latency and task processing deadlines. The learning model is constantly trained with the fresh data fed by the digital twins of the vehicles.

Data-driven routing is a modern approach to network routing where the data within the packets is taken into account for dispatching the packets as opposed to using the header information only. Reference [29] has designed a digital twin network architecture used for analysis and simulation of network traffic management decisions. This virtual model is constantly updated with real-time data and stores the entire history of the network up to the current time. They have proposed a learning-based data-driven routing algorithm that is trained by the real-time data provided by the digital twin in the training stage. Ultimately, in the deployment stage, the trained algorithm makes specialized routing decisions according to the current traffic status.

The 6th generation of mobile networks will feature edge intelligence and extremely low latency. To meet these demands, [7], [30] have designed a digital twin model of the wireless network helping in the mitigation of unreliable and long-distance communications between end users and edge servers. The digital twin model of the network already present in the edge servers assists a server in optimally placing the DTs of the mobile end users and migrating them to minimize the user-server communication latency.

Chapter 3

System Model and Problem Formulation

Centralized DT (Figure 1(a)) is a DT placement architecture where a digital twin interacts with an application on behalf of the physical system it is representing. The digital twin, placed somewhere between the PS and the application server, is periodically updated by the PS with its latest state changes. This is referred to as PS-DT synchronization. The DT, after processing the received updates, sends them to the application server which has requested information regarding the PS.

In this architecture, the direct communication between applications and real systems is replaced by two communication paths namely, a PS-DT path plus a DT-Application path. Since the DT is located closer to the application server, the latter will experience lower response delays from the DT compared to response latencies from the PS. Nevertheless, if the DT is located too close to the application, the PS-DT synchronization latency will start to increase which will degrade the freshness of information provided to the application.

As the number of physical devices increases, more DTs will populate the available servers (circles in Figure 2) and consume more computational resources of these servers.

Each server must be able to process the incoming updates of the hosted DTs in a timely manner so that the DTs will remain synchronized with their corresponding PS. On the other hand, the server should respond to application requests without too much latency. A real-life scenario for this system model is the case where the PSs are sensors installed around a manufacturing plant and a maintenance application, responsible for accident prevention, periodically requests fresh data from the sensors. In this thesis, the DT placement problem is investigated. Our objective is to minimize the maximum response delay experienced by the application subject to data age targets set by the application server over the information it receives.

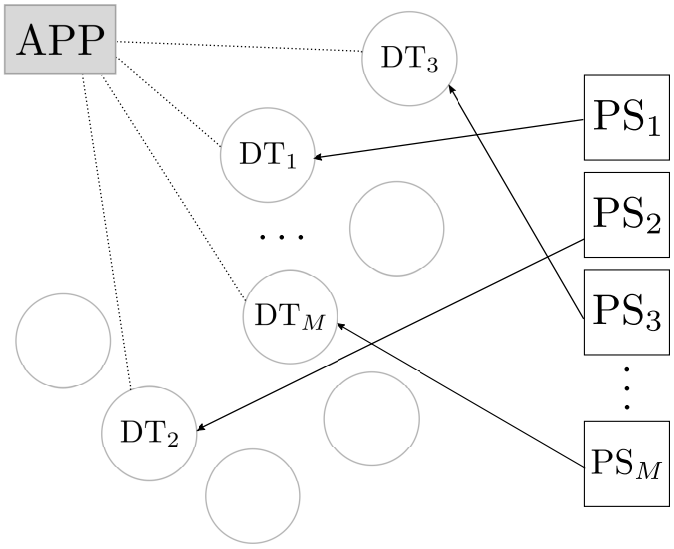


Figure 2: Multiple DTs providing information to an application on behalf of their corresponding PSs. Arrows and dashed lines are communication paths that may consist of one or more router hops.

3.1 System Model

The system model consists of a set of M physical systems $\mathcal{M} = \{PS_1, PS_2, \ldots, PS_M\}$, the set of N execution servers $\mathcal{N} = \{ES_1, ES_2, \ldots, ES_N\}$ that can host digital twins, and an application server (shown in Figure 2). The application server (AS) requires data input from the set of physical systems. Every PS is associated with a digital twin that

receives the data request of AS and responds to it using the information obtained from the PS. The digital twin of PS_m (DT_m) has to be placed in one execution server. We will denote DT_m placed in ES_n by the pair (m, n). A digital twin will impose communication costs to the network (for receiving updates from the PS and sending them to the application server) and computation costs to its hosting ES (for processing the received updates). Figure 3 demonstrates the timeline of interactions between a PS, its DT, and the AS.

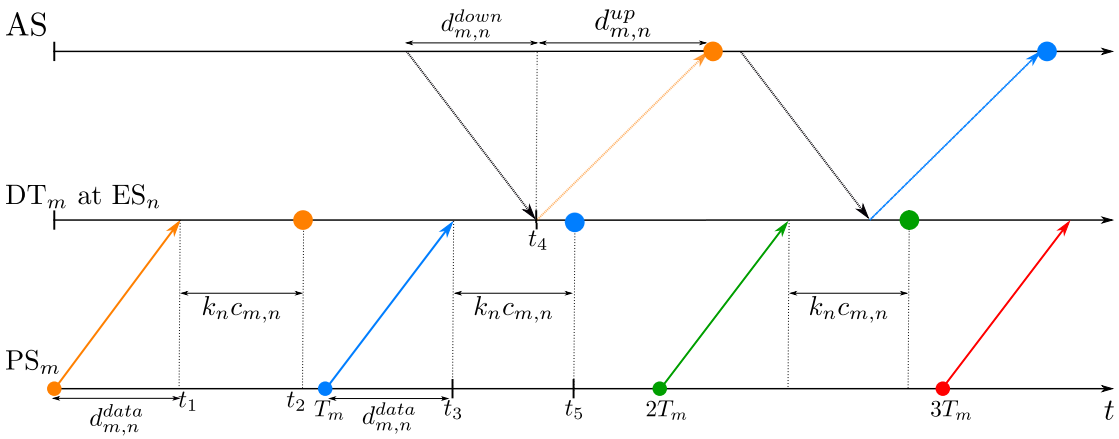


FIGURE 3: Timeline of PS-DT and DT-AS interactions.

At the beginning of the timeline, PS_m sends a state update to its DT at ES_n (indicated by the orange color in the Figure). The transmission delay of this update is denoted by $d^{data}_{m,n}$. At $t = t_1$, the update is available at ES_n in its raw form and needs to be processed by the DT. Assuming that the CPU frequency of ES_n and the required CPU cycles for processing the update are known, it will take $c_{m,n}$ seconds for the update to be processed. This is the case where only one DT rests inside the ES and is free to use the entire computing power of the server. If, however, there is more than one DT in an ES, we apply the *time-sharing* policy to maintain fairness among the hosted DTs.

The time-sharing policy dictates that once a digital twin has used one cycle of the server's processor, it has to wait for the other DTs to use one CPU cycle to be able to

use another cycle of the CPU. As a result of this policy, If there are k_n digital twins in ES_n, it will take at most $k_n c_{m,n}$ seconds for the CPU to finish processing the update of DT_m. [1] Back in the timeline, the update is fully processed at $t = t_2 = t_1 + k_n c_{m,n}$ and PS_m and its DT are now state-synchronized.

As the state of a physical system changes over time, the corresponding digital twin needs to get periodic updates to remain synchronized with the PS. In our system model, we assume that PS_m sends an update to DT_m every T_m seconds. Thus, as indicated in the timeline, the second, third, and fourth updates are sent at $t = T_m$, $t = 2T_m$, and $t = 3T_m$ respectively (shown by the colors blue, green, and red).

Suppose the application server sends an information request to the digital twin at a certain time. Once the request is made, it will take $d^{down}_{m,n}$ seconds for it to reach the edge server ($t = t_4$). As soon as the ES receives the request, it will start sending the last processed state information to the AS for a duration of $d^{up}_{m,n}$ seconds. In the timeline, ES_n receives the request at $t = t_4 < t_5$. Since the second update is still being processed, the server sends the state information of the previous state of the digital twin i.e. the orange update. The discussed system model is summarized in Figure 4.

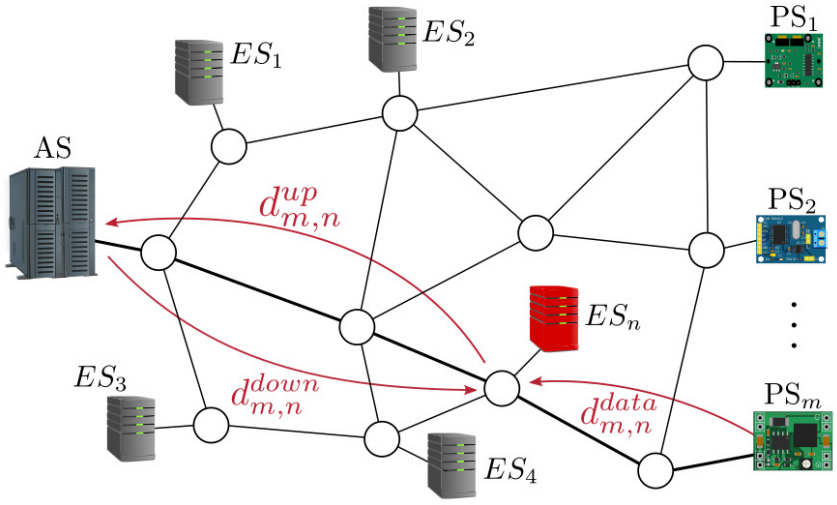


FIGURE 4: System model for digital twin placement problem.

[1] Note that this is an upper bound for the processing delay since we have implicitly assumed that DTs always have an update that is waiting to be processed.

It is seen from the timeline 3 that the state of a DT should be at most one state behind the state of its PS. For this constraint to hold, the incoming updates have to reach the hosting edge server and get processed before the next one arrives i.e.

$$d_{m,n}^{data} + k_n c_{m,n} \leq T_m \tag{3.1}$$

if DT_m is located at ES_n and there are k_n DTs at ES_n in total.

In our system model, a third-party application sends information requests to the digital twins that are residing on the edge servers. Naturally, the application expects to receive near real-time information from the physical devices. Therefore, while placing the digital twins, we have to account for the age of information [2] (AoI) that the application sees and try to keep it as small as possible. We assume that the application accepts data that is at most $A^\star$ seconds old. The worst-case age of information has to be less than this bound. According to the timeline in Figure 3, the worst-case age of information happens when the application's request for DT_m's data hits the edge server just before DT_m has finished processing the most recent update i.e. $t_4 < t_5$ and $t_5 - t_4 \approx 0$. The age of information for the orange data at $t = t_4 \approx t_5$ is $T_m + d_{m,n}^{data} + k_n c_{m,n}$. With an additional ES-AS transmission delay of $d_{m,n}^{up}$ seconds, the age of information constraint can be written as:

$$T_m + d_{m,n}^{data} + k_n c_{m,n} + d_{m,n}^{up} \leq A^\star \tag{3.2}$$

Before formulating the optimization problem, we will point out an assumption regarding the data transmission delays. Similar to assumptions made in [6], we assume that the transmission delays are independent of the placement strategy and the number

[2] Age of Information is defined as the elapsed time from the moment data is generated until it reaches the application server.

of DTs placed in each edge server. This is because the total communication load on the network imposed by the DTs is far less than the actual capacity of the underlying links that are connecting the edge servers.

3.2 Problem Formulation

The decision variables of our problem are defined as binary variables $X_{mn} \in \{0, 1\}$ for $m \in \{1, 2, \ldots, M\}$ and $n \in \{1, 2, \ldots, N\}$ where $X_{mn} = 1$ if DT_m is placed on ES_n and $X_{mn} = 0$ if not. We can write the number of digital twins that are hosted in ES_n as $k_n = \sum_{m=1}^{M} X_{mn}$. As a result, constraint (3.1) can be changed to

$$X_{mn}(d_{m,n}^{data} + c_{m,n} \sum_{m=1}^{M} X_{mn}) \leq T_m \tag{3.3}$$

and constraint (3.2) to

$$X_{mn}(T_m + d_{m,n}^{data} + c_{m,n} \sum_{m=1}^{M} X_{mn} + d_{m,n}^{up}) \leq A^{\star} \tag{3.4}$$

Our objective is to provide the requested information to the application server as soon as possible. The application will get data from DT_m at ES_n, $d_{m,n}^{down} + d_{m,n}^{up}$ seconds after it sends a request for it. Our approach is to minimize the maximum experienced delay by the AS among all the digital twins that the application requests data from; namely the optimal placement is

$$X^{\star} = \arg\min_{X} \max_{m} \{(d_{m,n}^{down} + d_{m,n}^{up}) X_{mn}\} \tag{3.5}$$

A common practice in min max objectives is to replace the $\max_m$ by a new decision variable, which we call τ, and add a constraint where every $(d^{down}_{m,n} + d^{up}_{m,n})X_{mn}$ is less than τ.

The digital twin placement problem is formulated as the following Integer Quadratic Program (IQP):

$$\min_{X,\tau} \quad \tau \tag{IQP}$$

$$\text{s.t.} \quad \sum_{n=1}^{N}(d^{down}_{m,n} + d^{up}_{m,n})X_{mn} \leq \tau, \quad \forall m \in \mathcal{M} \tag{3.6}$$

$$X_{mn}(d^{data}_{m,n} + c_{m,n}\sum_{k=1}^{M} X_{kn}) \leq T_m, \quad \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{3.7}$$

$$X_{mn}(T_m + d^{data}_{m,n} + c_{m,n}\sum_{k=1}^{M} X_{kn} + d^{up}_{m,n}) \leq A^{\star}, \quad \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{3.8}$$

$$\sum_{n=1}^{N} X_{mn} = 1, \quad \forall m \in \mathcal{M} \tag{3.9}$$

$$X_{mn} \in \{0, 1\}, \quad \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{3.10}$$

$$\tau \geq 0 \tag{3.11}$$

The LHS of constraint (3.6) is the application's experienced latency when requesting data from DT_m. By minimizing τ, we are minimizing the maximum of such latencies. Constraints (3.7) and (3.8) are the digital twin update constraints and application's data age constraints respectively. Constraint (3.9) implies that every digital twin has to be placed in exactly one edge server. This will also prevent the optimization to zero out the $\mathbf{X}$ matrix.

By re-writing constraints (3.7) and (3.8) in the following form

$$X_{mn}\sum_{k=1}^{M} X_{kn} \leq \frac{T_m - X_{mn}d^{data}_{m,n}}{c_{m,n}} \tag{3.12}$$

$$X_{mn}\sum_{k=1}^{M}X_{kn}\leq\frac{A^{\star}-X_{mn}(d_{m,n}^{data}+d_{m,n}^{up})}{c_{m,n}} \tag{3.13}$$

we obtain the same LHS for both constraints. The X_{mn} term can be removed from the RHS of both constraints because for $X_{mn}=0$ the constraints are always satisfied and we only need to take care of the case when $X_{mn}=1$. We can now merge the two constraints by defining a new RHS $u_{m,n}$:

$$u_{m,n}=\min\left\{\frac{T_m-d_{m,n}^{data}}{c_{m,n}},\frac{A^{\star}-(d_{m,n}^{data}+d_{m,n}^{up})}{c_{m,n}}\right\} \tag{3.14}$$

By applying the aforementioned changes the new formulation is

$$\min_{X,\tau}\quad\tau \tag{IQP'}$$

$$\text{s.t.}\quad\sum_{n=1}^{N}(d_{m,n}^{down}+d_{m,n}^{up})X_{mn}\leq\tau,\quad\forall m\in\mathcal{M} \tag{3.15}$$

$$X_{mn}\sum_{k=1}^{M}X_{kn}\leq u_{m,n},\quad\forall m\in\mathcal{M},n\in\mathcal{N} \tag{3.16}$$

$$\sum_{n=1}^{N}X_{mn}=1,\quad\forall m\in\mathcal{M} \tag{3.17}$$

$$X_{mn}\in\{0,1\},\quad\forall m\in\mathcal{M},n\in\mathcal{N} \tag{3.18}$$

$$\tau\geq 0 \tag{3.19}$$

In the rest of this chapter, we will prove the NP-completeness of (IQP'). By fixing the τ to a specific $\hat{\tau}$ value, all X_{mn} in (3.15) with coefficients $d_{m,n}^{down}+d_{m,n}^{up}>\hat{\tau}$ are forced to be 0 according to constraints (3.17) and (3.18). The τ variable can take any value from the $[d^{down}+d^{up}]_{M\times N}$ matrix. We define $D_{sorted}=\{\hat{\tau_1},\hat{\tau_2},\ldots,\tau_{\hat{MN}}\}$ as the sorted array of the elements of $[d^{down}+d^{up}]_{M\times N}$ matrix. By fixating $\tau=\hat{\tau}$ according to a binary search on D_{sorted}, we can simplify the problem of (IQP') to the question of the feasibility of constraints (3.16)-(3.18) i.e. at every stage of the binary search, if the

problem is feasible, we ignore the half with larger τ values and vice versa.

Every value of $\tau = \hat{\tau}$, in effect, defines a bipartite graph $G = (A, B, E)$ (shown in Figure 5) with nodes in A corresponding to DTs, the nodes in B corresponding to ESs, and an edge $(m, n) \in E$ only if $d^{down}_{m,n} + d^{up}_{m,n} \leq \hat{\tau}$. For every value of τ, the problem (IQP') is simplified to the feasibility problem of constraints (3.16)-(3.18) on a bipartite graph ($X_{mn} \coloneqq 0$ whenever $(m, n) \notin E$ and $X_{mn} \coloneqq 1$ whenever $(m, n) \in E$). The following theorem proves that this feasibility problem is NP-complete.

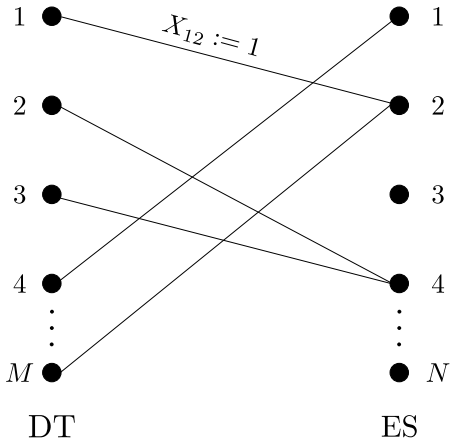


Figure 5: A bipartite Graph $G = (A, B, E)$.

Theorem 1. *Determining the feasibility of* (3.16)-(3.18) *on a bipartite graph is an NP-complete problem.*

Proof. The problem is clearly in NP since given a $\{0, 1\}$-assignment for X_{mn} variables, the feasibility of constraints (3.16)-(3.18) can be checked in polynomial time.

We reduce the Boolean Satisfiability problem (also Satisfiability or SAT) to our problem. SAT problem asks whether there is a truth assignment that satisfies a given CNF[3] formula. Given a CNF formula with n variables and m clauses, we construct a bipartite graph as follows:

The left-side set of nodes A consists of $n + m$ nodes, one for each variable or clause.

[3]Conjunctive Normal Form or Clausal Normal Form

On the right-side set of nodes B there are $2n$ nodes, one pair of nodes for every pair of literals $x_i, \bar{x}_i$ corresponding to i-th variable. For every clause l, there is an edge between $l \in A$ and the node of every literal used by l in B. For example, for clause $l = (x_2 \vee \bar{x}_5 \vee x_8)$ there are edges (l, x_2), $(l, \bar{x}_5)$, (l, x_8). For each such edge (m, n) we set $u_{m,n} := \infty$. Also, for the i-th variable node in A, we add edges $(i, x_i), (i, \bar{x}_i)$ with $u_{i,x_i} = u_{i,\bar{x}_i} := 1$. By construction, if $X_{i,x_i} = 1$, then $X_{l,x_i} = 0$ for any clause l that uses literal x_i, due to (3.16) for $m = i, n = x_i$ i.e. only variables $X_{k,\bar{x}_i}$ will be allowed to take value 1 where clauses k use literal $\bar{x}_i$; the case $X_{i,\bar{x}_i} = 1$ is symmetric.

Now it is easy to see that the given CNF formula is satisfiable iff there is a $\{0, 1\}$-assignment to variables $\mathbf{X}$ that also satisfies (3.16)-(3.18). If the formula is satisfiable, set $X_{i,x_i} = 1$ and $X_{i,\bar{x}_i} = 0$ if $x_i = 0$ or $X_{i,x_i} = 0$ and $X_{i,\bar{x}_i} = 1$ if $x_i = 1$. Furthermore, each clause l must contain a literal x_i or $\bar{x}_i$ that is set to 1, thus we can set $X_{l,x_i} = 1$ or $X_{l,\bar{x}_i} = 1$ without violating any of the constraints. We set all other variables $X_{m,n} := 0$. It is easy to verify that this assignment satisfies all constraints (3.16)-(3.18). Conversely, if there is a value assignment to variables $X_{m,n}$ that satisfies (3.16)-(3.18), then this assignment forces $X_{i,x_i} = 1, X_{i,\bar{x}_i} = 0$ or $X_{i,x_i} = 0, X_{i,\bar{x}_i} = 1$ for each i-th variable; we translate this assignment to $x_i = 0$ or $x_i = 1$ respectively. The assignment of *each* clause node l to exactly one node of its literal(s) in the bipartite graph is consistent with our truth assignment and satisfies each clause. □

As a result of Theorem 1, we do not expect that there is a polynomial-time algorithm that solves (IQP').

Chapter 4

Proposed Solution and Algorithms

In this chapter, we first simplify problem (IQP') to a relaxed semi-definite program (SDP) and then propose approximation algorithms to round the fractional solution of the SDP to integral binary values.

4.1 Problem Simplification and Relaxation

We will linearize problem (IQP') by defining a new binary variable $Z^n_{km} \in \{0, 1\}$ that will replace the product $X_{kn}X_{mn}$. The following linear constraints are a valid replacement for the quadratic constraint $Z^n_{km} = X_{kn}X_{mn}$:

$$
\begin{aligned}
&Z^n_{km} = Z^n_{mk}, \quad \forall k, m \in \mathcal{M}, n \in \mathcal{N} \\
&Z^n_{mm} = X_{mn}, \quad \forall m \in \mathcal{M}, n \in \mathcal{N} \\
&Z^n_{km} \leq X_{mn}, \quad \forall k, m \in \mathcal{M}, n \in \mathcal{N} \\
&X_{mn} + X_{kn} - Z^n_{km} \leq 1, \quad \forall k, m \in \mathcal{M}, n \in \mathcal{N}
\end{aligned}
$$

By applying the change of variables to (IQP') the new formulation becomes:

$$\min_{X,Z,\tau} \quad \tau \tag{SDP}$$

$$\text{s.t.} \quad \sum_n (d^{down}_{mn} + d^{up}_{mn}) X_{mn} \leq \tau, \quad \forall m \in \mathcal{M} \tag{4.1}$$

$$\sum_k Z^n_{km} \leq u_{m,n}, \quad \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{4.2}$$

$$\sum_n X_{mn} = 1, \quad \forall m \in \mathcal{M} \tag{4.3}$$

$$Z^n_{km} = Z^n_{mk}, \quad \forall k, m \in \mathcal{M}, n \in \mathcal{N} \tag{4.4}$$

$$Z^n_{mm} = X_{mn}, \quad \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{4.5}$$

$$Z^n_{km} \leq X_{mn}, \quad \forall k, m \in \mathcal{M}, n \in \mathcal{N} \tag{4.6}$$

$$X_{mn} + X_{kn} - Z^n_{km} \leq 1, \quad \forall k, m \in \mathcal{M}, n \in \mathcal{N} \tag{4.7}$$

$$X_{mn}, Z^n_{mk} \in \{0, 1\}, \quad \forall k, m \in \mathcal{M}, n \in \mathcal{N} \tag{4.8}$$

$$\tau \geq 0 \tag{4.9}$$

$$\mathbf{Z}^n \in \mathcal{PSD}, \quad \forall n \in \mathcal{N} \tag{4.10}$$

A feasible solution to (SDP) will set $Z^n_{km} = X_{kn} X_{mn}, \forall k, m, n$ the $\mathbf{X}$ matrix of which is a feasible solution to (IQP'). Matrices $\mathbf{Z}^n$ are positive semi-definite (PSD) for all n since $\mathbf{Z}^n = \mathbf{X}^n (\mathbf{X}^n)^T$, where $\mathbf{X}^n$ is the n-th column of matrix $\mathbf{X} = [x_{mn}]$. Therefore, we can add the constraint

$$\mathbf{Z}^n \in \mathcal{PSD}, \quad \forall n \in \mathcal{N} \tag{4.11}$$

which will turn the linearized problem into a semi-definite program (SDP).

We apply a relaxation of binary integer variables by replacing (4.8) with

$$X_{mn}, Z^n_{km} \geq 0, \quad \forall k, m \in \mathcal{M}, n \in \mathcal{N} \tag{4.12}$$

The problem (SDP) now becomes a convex SDP (note that constraints $X_{mn} \leq 1$ and $Z^n_{km} \leq 1$ are implied by (4.3) and (4.6)) and can be solved in polynomial time by common SDP solvers. The final *Relaxed SDP* problem is summarized as follows:

$$\min_{X,Z,\tau} \quad \tau \qquad \text{(Relaxed SDP)}$$

$$\text{s.t.} \quad (4.1) - (4.7), (4.9) \tag{4.13}$$

$$X_{mn} \geq 0, \quad \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{4.14}$$

$$Z^n_{km} \geq 0, \quad \forall k, m \in \mathcal{M}, n \in \mathcal{N} \tag{4.15}$$

$$\mathbf{Z}^n \in \mathcal{PSD}, \quad \forall n \in \mathcal{N} \tag{4.16}$$

If (Relaxed SDP) is infeasible for a given input, this implies that the original (IQP') problem is infeasible as well. However, the infeasibility of input on (IQP') does not necessarily mean that (Relaxed SDP) is infeasible for that input.

4.2 Solution Method

The formulation of problems (IQP') and (Relaxed SDP) treats (3.16) and (4.2) respectively as a *hard* constraint, i.e., they would be satisfied if at least one feasible solution was available. In real-world systems, considering the dynamic nature of the system parameters that correspond to the input of our problem, a feasible solution may not exist for all parameter settings. Furthermore, checking the feasibility of the problem is itself NP-complete as proven in Theorem 1. Therefore, we propose polynomial-time approximation algorithms that return approximate solutions in terms of the objective τ and violation of constraint (4.2).

The algorithms give precedence either to achieving a better objective value τ (to the detriment of constraint (4.2)) or to attaining a smaller violation of constraint (4.2) (to the detriment of the objective τ). We will later test the algorithms on instances where

problem (IQP') is feasible, even though the algorithms provide an approximate solution to the infeasible instances of problem (IQP') and only require Relaxed SDP problem to be feasible.

4.2.1 Rounding Algorithm

We start by describing Algorithm 1 which rounds a fractional solution of the Relaxed SDP problem to an integral assignment of DTs to ESs. This rounding subroutine will be the main component of the algorithms we propose for solving the problem or detecting its infeasibility (Algorithms 2 & 3).

After obtaining the fractional solution $\mathbf{X}, \mathbf{Z}$ of the Relaxed SDP problem, Algorithm 1 is used to round it to an integral solution. In lines 2-6, all the integral X_{mn}'s from the solution to the Relaxed SDP problem are fixed, i.e., DT_m is assigned to ES_n if $X_{m,n} = 1$ and it will never be assigned to ES_n if $X_{m,n} = 0$. After the `for`-loop, set $\mathcal{SM}$ contains the DTs that are still fractionally assigned to different ESs.

In line 8, while $\mathcal{SM} \neq \emptyset$, the algorithm picks DT_m from $\mathcal{SM}$ and a pair of ESs n_1, n_2 with $0 < X_{m,n_1}, X_{m,n_2} < 1$ according to different edge-pair selection algorithms that will be described in subsection 4.2.2.

In lines 9-13, we set $X_{m,n_1} := X_{m,n_1} + X_{m,n_2}$ and $X_{m,n_2} := 0$ and remove DT_m from the set of fractional DTs $\mathcal{SM}$ if $X_{mn_1} = 1$. Note that this update may violate some constraints, therefore, the algorithm goes through a series of operations to satisfy the violated constraints again. This includes increasing T_m and $A^\star$ to satisfy constraint (4.2) in lines 14-28 and adjusting the $Z^n_{k,m}$ values based on the updated $X_{m,n}$'s to satisfy constraints (4.6) and (4.7) in lines 29-36. Note that fixing constraints (4.7) after constraints (4.6) ensures that the latter will still be satisfied.

Algorithm 1 Rounding Algorithm

Require: Fractional solution $\mathbf{X}, \mathbf{Z} \geq \mathbf{0}$ of (Relaxed SDP)

$\mathcal{SM} := \mathcal{M}$
for all $\mathtt{m} \in \mathcal{SM}$ **do**
  if $X_{m,n} \in \{0,1\}$ for some n **then**
    $\mathcal{SM} := \mathcal{SM} \setminus \{m\}$
  end if
end for
while $\mathcal{SM} \neq \emptyset$ **do**
  Select(m, n_1, n_2) ▷ Edge-pair selection algorithms
  $X_{m,n_1} := X_{m,n_1} + X_{m,n_2}$
  $X_{m,n_2} := 0$
  if $X_{m,n_1} = 1$ **then**
    $\mathcal{SM} := \mathcal{SM} \setminus \{m\}$
  end if
  for all $m \in \mathcal{M}, n \in \mathcal{N}$ **do**
    $u^{\star}_{m,n} := u_{m,n}$
    if constraint (4.2) is violated **then**
      $u_A := \frac{A^{\star} - (T_m + d^{data}_{m,n} + d^{up}_{m,n})}{c_{m,n}}$
      $u_T := \frac{T_m - d^{data}_{m,n}}{c_{m,n}}$
      if $u_T < LHS < u_A$ **then** ▷ LHS of (4.2)
        Increase T_m until $u_T = LHS$
      else if $u_A < LHS < u_T$ **then**
        Increase $A^{\star}$ until $u_A = LHS$
      else if $u_T, u_A < LHS$ **then**
        Increase T_m and $A^{\star}$ until $u_A = u_T = LHS$
      end if
      $u_{m,n} = \min\{u_A, u_T\}$
    end if.
  end for
  for all $k, m \in \mathcal{M}, n \in \mathcal{N}$ **do**
    if constraint (4.6) is violated **then**
      $Z^n_{k,m}, Z^n_{m,k} := X_{m,n}$
    end if
    if constraint (4.7) is violated **then**
      $Z^n_{k,m}, Z^n_{m,k} := X_{m,n} + X_{k,n} - 1$
    end if
  end for
end while
Output: $\mathbf{X}$, $\mathbf{Z}$, and Δu

Line 15 records the original $u_{m,n}$ as $u^{\star}_{m,n}$ and line 26 records the updated $u_{m,n}$. Let

$$\Delta u = \max_{m,n} \frac{u_{m,n} - u^{\star}_{m,n}}{u^{\star}_{m,n}} \tag{4.17}$$

be the maximum violation of $u_{m,n}$ due to the rounding. The algorithm outputs the rounded $\mathbf{X}$, $\mathbf{Z}$, and Δu.

4.2.2 Edge-pair Selection Algorithms

We use five different methods for the choice of m, n_1, n_2 in line 8 of Algorithm 1.

- **Random Selection:** We pick a DT m from $\mathcal{SM}$ and two ESs n_1 and n_2 from $\mathcal{N}$ with $0 < X_{m,n_1}, X_{m,n_2} < 1$ uniformly at random.

- **X-Congestion:** The metric $\sum_m X_{mn}$ is used as a measure of the *congestion* of ES_n. Let $n_2 = \arg\max_n \sum_m X_{mn}$ (n_2 is the most congested ES) and $m = \arg\max_k X_{kn_2}$. Set $n_1 = \arg\min_n X_{m,n}$ (we break ties arbitrarily).

- **Z-Congestion:** The metric $\sum_{k,m} Z^n_{km}$ is used as a measure of the congestion of ES_n. Let $n_2 = \arg\max_n \sum_{k,m} Z^n_{km}$ and $n_1 = \arg\min_n \sum_{k,m} Z^n_{km}$. m is selected as before.

- **Constraint Slack SDP:** Let $(m, n_2) = \arg\min_{m,n}\{u_{m,n}\}$ i.e. (m, n_2) is the DT-ES pair for which the slack of constraint (4.2) is minimum. Note that if any of these constraints are tight, the minimum slack is 0. Set $n_1 = \arg\max_n u_{m,n}$, i.e., for the chosen m, n_1 is the ES that has the maximum slack among all the constraints (4.2). The intuition behind this selection of m, n_1, and n_2 is that the algorithm is trying to take away assignment "weight" from tight (or near tight) constraints and assign it to constraints with lots of slack.

- **Constraint Slack QP:** Consists of the same operations as in **Constraint Slack SDP** only now we consider the slack of constraints (3.16).

Note that every iteration of the main loop of Algorithm 1 (lines 7-37) rounds at least one of the $O(MN)$ variables and each iteration takes $O(MN)$ time for an overall running time of $O(M^2N^2)$.

4.2.3 Approximation Algorithms

Algorithm 1 is used as a subroutine to develop the following approximation algorithms that work towards restricting deviations of fractional u or τ (Algorithms 2 and 3 respectively) compared to the optimal solution.

Algorithm 2 Finding solution with sub-ϵ_u violation

Require: $D_{sorted} = \{\hat{\tau}_1, \hat{\tau}_2, \ldots, \hat{\tau}_{MN}\}, \epsilon_u$
1: $\hat{\tau}_f$ = min in D_{sorted} s.t. Relaxed SDP is feasible ▷ Binary search in D_{sorted}
2: $\hat{\tau}_s$ = min in $\{\hat{\tau}_f, \hat{\tau}_{f+1}, \ldots, \hat{\tau}_{MN}\}$ s.t. ▷ Binary search
- X_f, Z_f = solution of Relaxed SDP problem with $\tau = \hat{\tau}_s$
- $X, Z, \Delta u$ = Algorithm 1(X_f, Z_f)
- $\Delta u \leq \epsilon_u$

3: **if** no $\hat{\tau}_s$ is found **then**
4: return INFEASIBLE
5: **else**
6: return **X**
7: **end if**

Algorithm 3 Finding solution with sub-ϵ_τ violation

Require: $D_{sorted} = \{\hat{\tau}_1, \hat{\tau}_2, \ldots, \hat{\tau}_{MN}\}, \epsilon_\tau$
1: τ_{opt}^{SP} = fractional optimum of Relaxed SDP
2: Find max s such that $\hat{\tau}_s \leq \tau_{opt}^{SP}(1 + \epsilon_\tau)$
3: **if** no $\hat{\tau}_s$ is found **then**
4: return INFEASIBLE
5: **end if**
6: X_f, Z_f = solution of Relaxed SDP problem with $\tau = \hat{\tau}_s$
7: $X, Z, \Delta u$ = Algorithm 1(X_f, Z_f)
8: return **X**

Let τ_{opt}^{QP} be the optimum τ of (IQP'), therefore, optimum τ of (SDP) as well. We observe that $\tau_{opt}^{QP} \in \{d_{m,n}^{up} + d_{m,n}^{down} : m \in \mathcal{M}, n \in \mathcal{N}\}$. Each of these MN possible values for τ_{opt}^{QP} corresponds to a restriction of the set of possible assignments of DTs to ESs. As previously defined, $D_{sorted} = \{\hat{\tau}_1, \hat{\tau}_2, \ldots, \hat{\tau}_{MN}\}$ is the sorted list of values $\{d_{m,n}^{up} + d_{m,n}^{down} : m \in \mathcal{M}, n \in \mathcal{N}\}$ in ascending order (note that if there are repetitions of values for different m, n combinations then $|D_{sorted}| < MN$, nevertheless, for the clarity of our presentation, we will assume that all these values are distinct). By fixating $\tau := \hat{\tau}_s \in D_{sorted}$ Relaxed SDP problem becomes a *feasibility* problem as follows: In order to satisfy (4.1), $X_{mn} = 0$ has to be set for all m and n with $d_{m,n}^{up} + d_{m,n}^{down} > \hat{\tau}_s$. Therefore, the following constraints are added:

$$X_{mn} = 0, \quad \forall m, n : d_{m,n}^{up} + d_{m,n}^{down} \in \{\hat{\tau}_{s+1}, \ldots, \hat{\tau}_{MN}\} \tag{4.18}$$

and are considered together with constraints (4.2)-(4.7) and (4.14)-(4.16). A feasible (and fractional) solution to this feasibility problem can be obtained in polynomial time. If the problem is infeasible, we proceed with a smaller value of τ. Recall that we have assumed that the Relaxed SDP is feasible, thus there is at least one value of τ for which we will obtain a fractional feasible solution.

In Algorithm 2, binary search is used in order to discover the smallest $\hat{\tau}_f \in D_{sorted}$ that maintains the feasibility of the Relaxed SDP problem. This is done after solving at most $O(\log(MN))$ SDPs. Note that $\hat{\tau}_f \leq \tau_{opt}^{QP}$, since the first is the fractional optimum and the latter is the integral optimum. A second binary search is applied to the set $\{\hat{\tau}_f, \hat{\tau}_{f+1}, \ldots, \hat{\tau}_{MN}\}$ in order to find the smallest $\hat{\tau}_s$ for which $\Delta u \leq \epsilon_u$ when the rounding of Algorithm 1 is applied (ϵ_u is the constraint violation tolerance). If no such solution is found for the given tolerance, the algorithm will return infeasible.

In Algorithm 2, we were aiming to find a solution with sub-ϵ_u constraint violation. Similarly, an ϵ_τ bound can be applied to the objective τ as follows: Let

$$\Delta\tau(x) = \frac{x - \tau_{opt}^{SP}}{\tau_{opt}^{SP}} \tag{4.19}$$

where τ_{opt}^{SP} is the (fractional) optimum of the Relaxed SDP. After calculating τ_{opt}^{SP}, algorithm 3 chooses the largest $\hat{\tau}_s$ from the set D_{sorted} with $\Delta\tau(\hat{\tau}_s) \leq \epsilon_\tau$. The Relaxed SDP problem for $\tau = \hat{\tau}_s$ is solved and the fractional solution is rounded using Algorithm 1. If no such $\hat{\tau}_s$ exists, the algorithm returns infeasibility. If $\mathbf{X}$ is the returned solution matrix by algorithms 2 and 3, we denote the obtained τ from $\mathbf{X}$ by $\hat{\tau}$ and define

$$\Delta\tau' = \frac{\hat{\tau} - \tau_{opt}^{QP}}{\tau_{opt}^{QP}} \tag{4.20}$$

where τ_{opt}^{QP} is the optimal objective value of IQP'.

Note that finding $\Delta\tau'$ requires the solution of the original NP-complete problem IQP' and will be used only in the simulations and algorithms do not rely on it in their operations. We have used the MOSEK optimization toolbox to solve IQP' and Relaxed SDP in our simulations.

Chapter 5

Simulation Results and Analysis

5.1 Simulation Setup

In this chapter, we evaluate the performance of our proposed algorithms via computer simulation. The algorithms are implemented using the five selection methods described in subsection 4.2.2. Since the solutions involve different relaxations of the constraints, the performance comparisons include the resulting constraint violation. Three sets of computer simulations were done to examine the performance of the algorithms from different perspectives. In the presented figures, each point represents an average of 50 simulation runs. Since constraint (3.16) consists of $M \times N$ separate constraints, at each simulation run, the maximum violation of that set is used in the averaging.

In the experiments, we will assume that the ESs are categorized into three groups: Group 1 which is nearest to PSs and furthest from the AS, Group 2 which is at a medium distance from the PSs and AS (compared to the other two groups), and Group 3 that is nearest to AS and furthest from the PSs. The delays for each group are defined accordingly:

- PS-ES delays d^{data} are uniformly distributed in ranges of $[5\text{ms}, 10\text{ms}]$, $[12.5\text{ms}, 17.5\text{ms}]$, and $[20\text{ms}, 25\text{ms}]$ for groups 1, 2, and 3 respectively.

- Application request delays d^{down} are uniformly distributed in ranges of $[0.7\text{ms}, 1.1\text{ms}]$, $[0.4\text{ms}, 0.8\text{ms}]$, and $[0.1\text{ms}, 0.5\text{ms}]$ for groups 1, 2, and 3 respectively.

- ES-AS response delays d^{up} are uniformly distributed in ranges of $[16\text{ms}, 20\text{ms}]$, $[10\text{ms}, 14\text{ms}]$, and $[4\text{ms}, 8\text{ms}]$ for groups 1, 2, and 3 respectively.

These ranges are chosen based on the fact that the network backbone of 5G is capable of supporting bit rates ranging from 5 Gbits/s up to 10 Gbits/s under stable and uncongested network conditions and some link processing overheads [31].

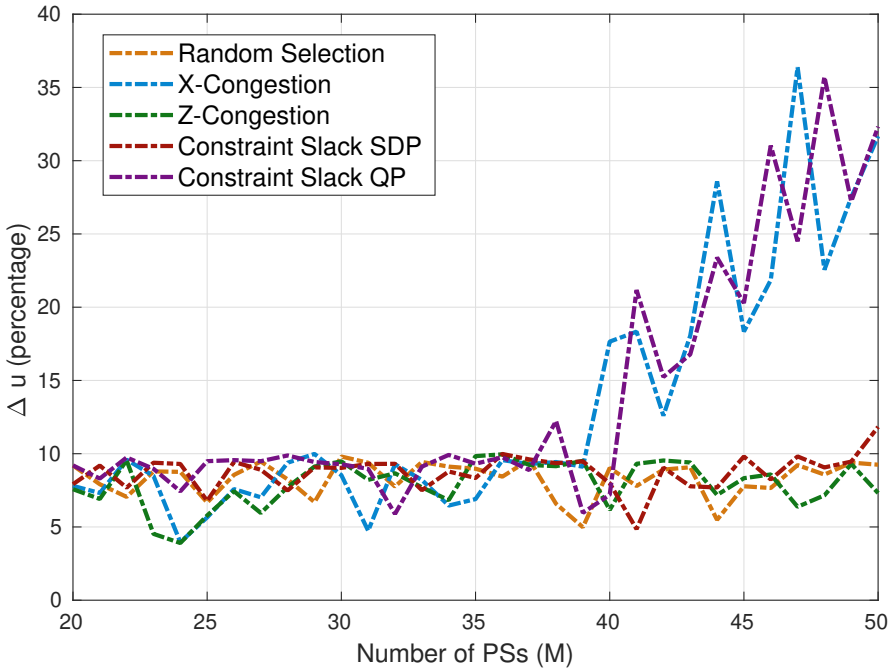
5.2 Simulation Results

5.2.1 Simulation Set 1

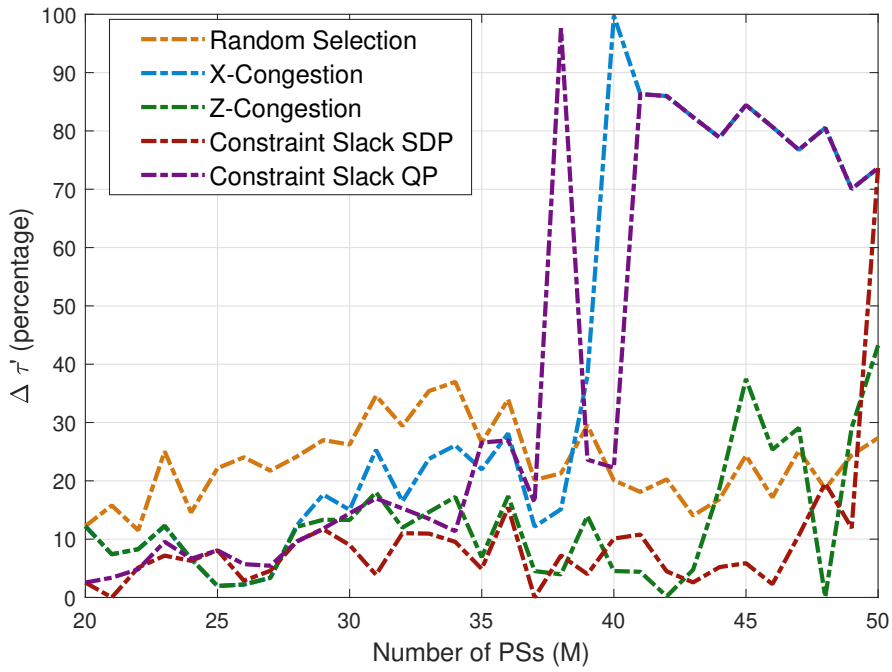
In the first set of simulations, 2 ESs are assigned to group 1, 4 ESs to group 2, and 2 ESs to group 3. The size of data sent by a PS to its DT at each data update cycle is 25MB, the data size sent from a DT to the application server is 20MB, and the size of the application's request is 1MB [32]. Additionally, T_m is uniformly distributed in the range $[60\text{ms}, 80\text{ms}]$ for each PS and application's data age target $A^\star$ is 200ms.

Algorithm 2 was run by varying the number of PSs. The results are plotted in figures 6(a) and 6(b) for $\epsilon_u = 10\%$ and figures 7(a) and 7(b) for $\epsilon_u = 5\%$. Figure 6(a) shows that all of the selection methods can keep Δu below ϵ_u up to $M = 40$ while Z-Congestion, Constraint Slack SDP and Random Selection can go up to $M = 50$. Figure 6(b) demonstrates that Z-Congestion and Constraint Slack SDP are able to keep $\Delta\tau'$ below 20%, unlike the other three methods.

The same trend can be seen in figures 7(a) and 7(b) as well. Figures 6(b) and 7(b) show that as M gets large, the selection methods offer larger $\Delta\tau'$ values ($M \in [45, 50]$) for a fixed ϵ_u boundary on constraint violation. We can also see that the case with $\epsilon_u = 10\%$ (figure 6(b)) provides smaller $\Delta\tau'$ compared to the case with $\epsilon_u = 5\%$ (figure 7(b)) which matches the trade-off between ϵ_u and $\Delta\tau'$.

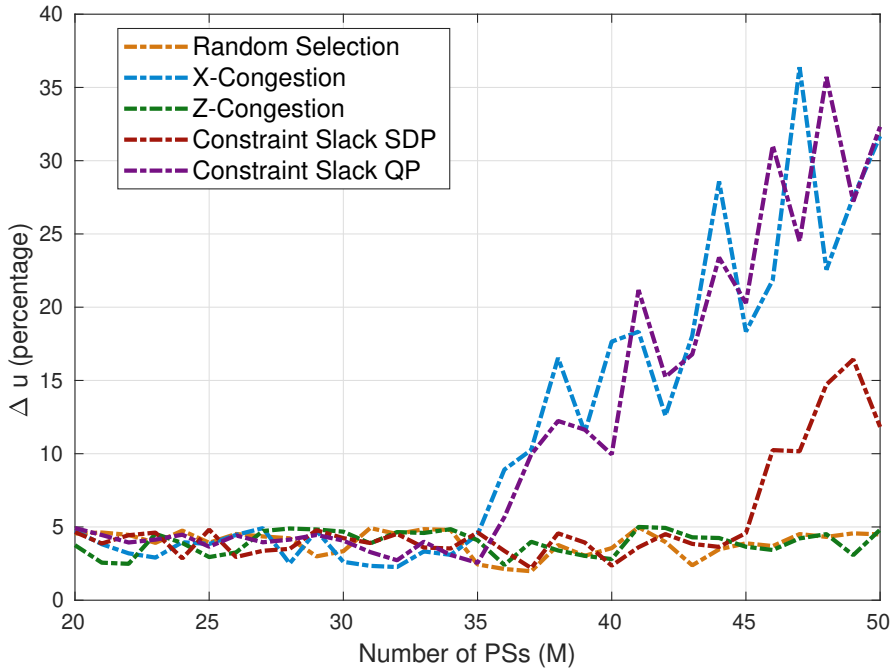


(A) Δu versus M

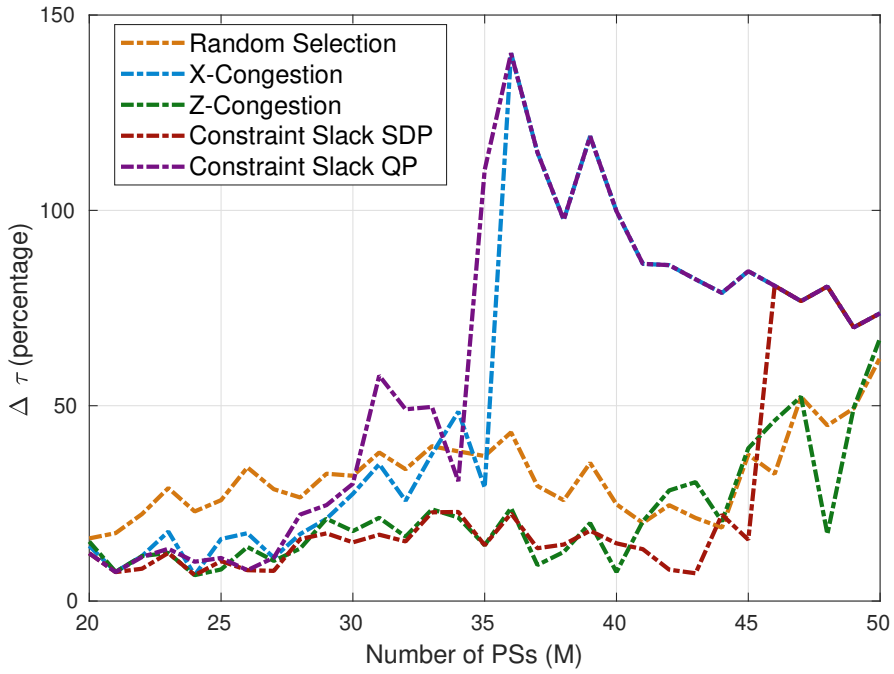


(B) $\Delta\tau'$ versus M

FIGURE 6: Performance of Algorithm 2, $\epsilon_u = 10\%$.



(A) Δu versus M



(B) $\Delta\tau'$ versus M

FIGURE 7: Performance of Algorithm 2, $\epsilon_u = 5\%$.

Algorithm 3 was tested for $\epsilon_\tau \in [0, 60\%]$ in figure 8. As expected, constraint violation Δu is maximum at $\epsilon_\tau = 0\%$ and starts to decrease as ϵ_τ increases. This trade-off between ϵ_τ and Δu is well reflected by Z-Congestion, Constraint Slack SDP, and Random Selection methods.

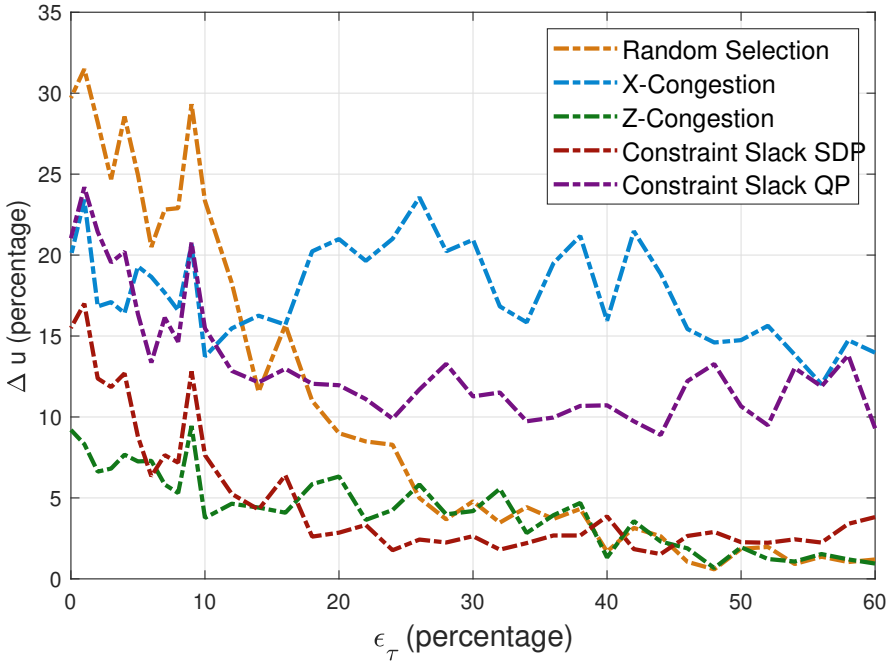


FIGURE 8: Performance of Algorithm 3 ($M = 40$).

Based on figures 6-8, we observe that X-Congestion and Constraint Slack QP show poor performance compared to the other methods with Random Selection being able to obtain smaller constraint violations than the latter two. Z-Congestion and Constraint Slack SDP are the two best-performing methods among our selection algorithms that consistently reach smaller τ violations for a fixed ϵ_u and smaller constraint violations for a fixed ϵ_τ. This can be explained by pointing out the fact that the linear operations of (9) in Algorithm 1 are applied to quadratic constraints in X-Congestion and Constraint Slack QP unlike the case for Z-Congestion and Constraint Slack SDP. Since $X_{mn} \leq 1$, under a quadratic term, ΔX_{mn} is significantly reduced and does not convey the same effect as in Z-Congestion and Constraint Slack SDP which use linear constraints.

5.2.2 Simulation Set 2

To further investigate the performance of the rounding methods, we ran another set of simulations. The network topology remains the same as in the previous set of simulations. Instead of specifying the values of T_m and $A^\star$, $u_{m,n}$ values are randomly generated according to the distribution in figure 9 which emulates a normal distribution. The results of running Algorithm 2 and 3 on the generated instances are plotted in figures 10(a) and 10(b). Similar to the results of simulation set 1, Z-Congestion gives consistently lower Δu values compared to the other rounding methods.

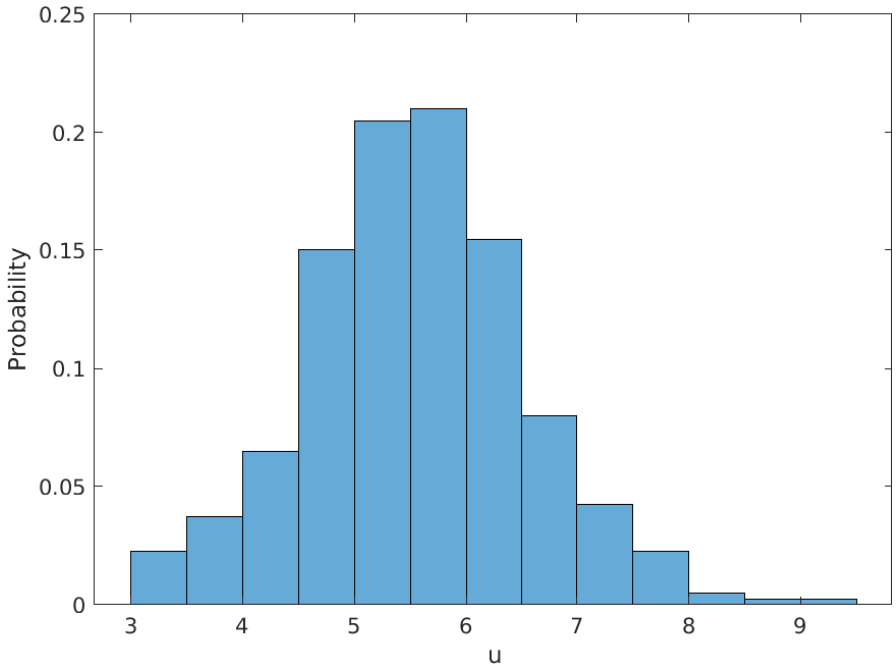
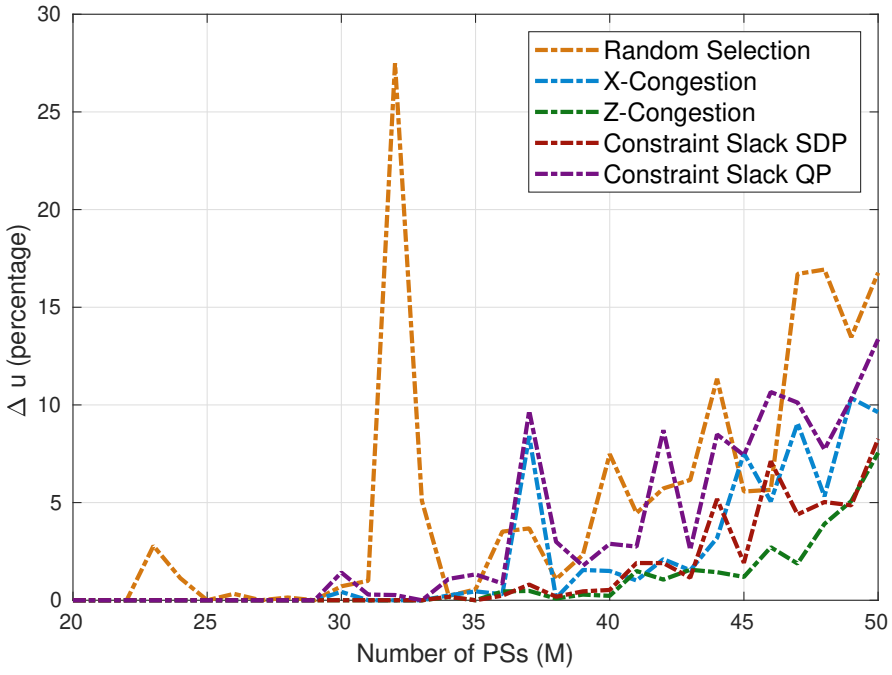
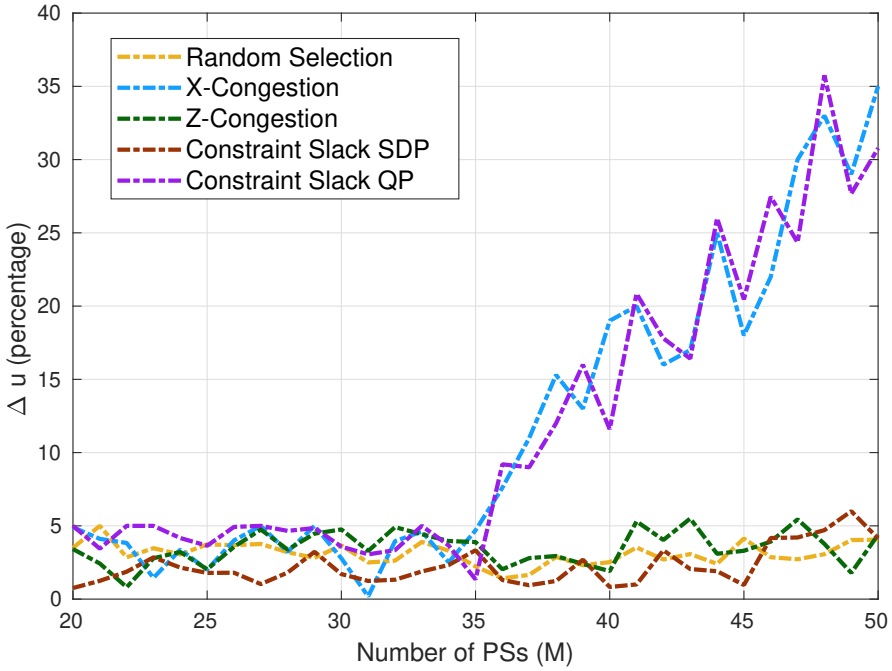


Figure 9: Probability distribution of u.



(A) Algorithm 2 ($\epsilon_\tau = 5\%$)



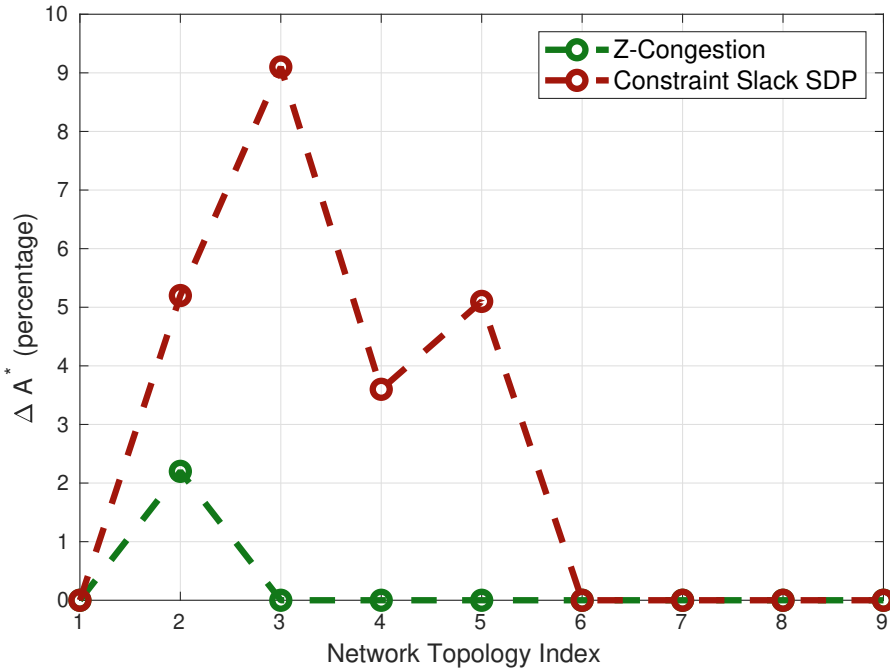
(B) Algorithm 3 ($\epsilon_u = 5\%$)

FIGURE 10: Constraint violation of selection algorithms over a non-uniform distribution of u.

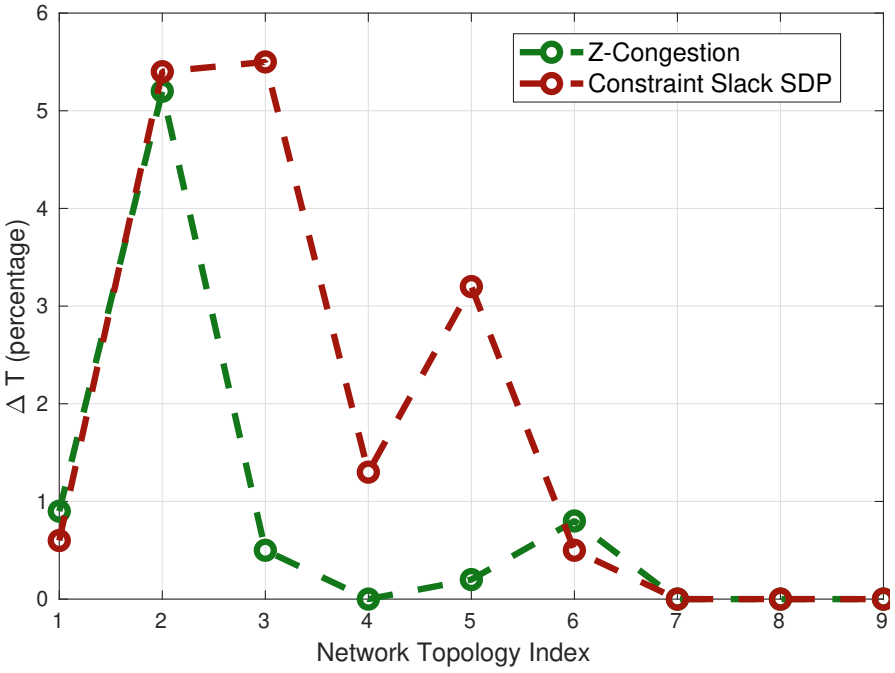
5.2.3 Simulation Set 3

Previous simulation results indicated that Z-Congestion and Constraint Slack SDP are superior to the others. Therefore, in the last set of simulations, we compare the performance of Z-Congestion and Constraint Slack SDP on the same network model as before but with a different number of ESs at each set. More specifically, let s_1, s_2, and s_3 be the number of ESs in groups 1, 2, and 3, respectively. We consider the following (s_1, s_2, s_3) tuple combinations indexed from 1 to 9: (2, 2, 4), (4, 2, 2), (2, 4, 2), (2, 0, 6), (6, 0, 2), (4, 0, 4), (0, 0, 8), (0, 8, 0), and (8, 0, 0). The performance results after running Algorithm 3 are given in figures 11 and 12 for $M = 30$ and 40, respectively. The figures contain $\Delta A^\star$ and ΔT_m violations that are calculated by comparing the original values of $A^\star$ and T_m with the ones obtained from the output solution of the algorithms (note that $\Delta T = \max_{m \in M} \Delta T_m$).

As shown in the figures, Z-Congestion consistently exhibits lower constraint violation than Constraint Slack SDP because Z-congestion achieves better load-balancing among the ESs. This happens for the reason that the constraint slacks used by Constraint Slack SDP depend on the server characteristics and parameters, while Z-Congestion ignores them and takes into account solely the load of the DTs on the ESs. We observe that if the DT assignment decisions rely on both server characteristics and server load, as done by Constraint Slack SDP, the heuristic can be misled into decisions that leave the ESs unbalanced. Clearly, if all ESs share the same amount of resources and characteristics, the two heuristics would have the same performance. This, however, is not the case in general and in our simulations.

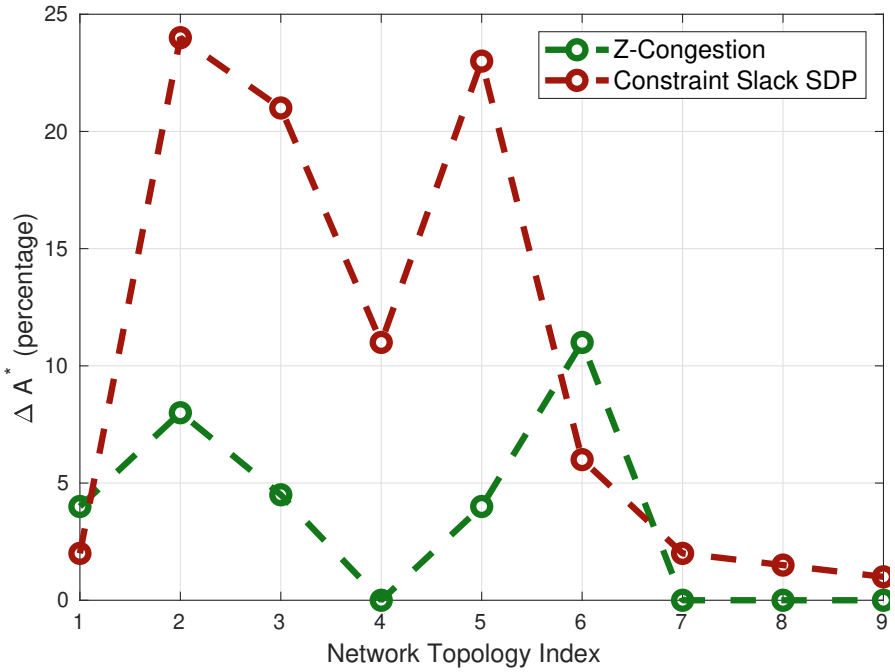


(A) $\Delta A^{\star}$

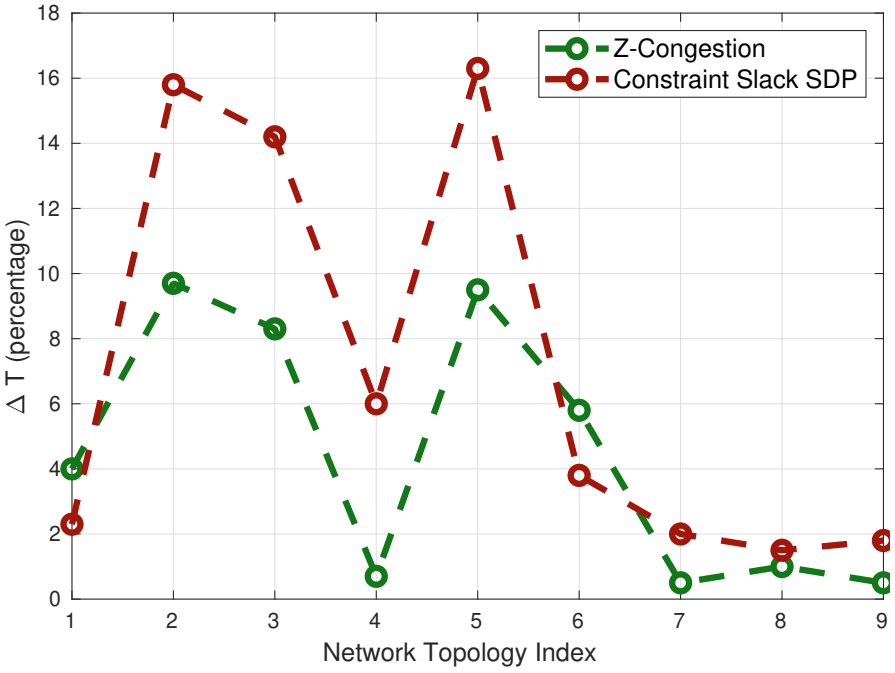


(B) ΔT

FIGURE 11: Performance of Constraint Slack SDP and Z-Congestion for $M = 30$ ($\epsilon_{\tau} = 5\%$).



(A) $\Delta A^{\star}$



(B) ΔT

FIGURE 12: Performance of Constraint Slack SDP and Z-Congestion for $M = 40$ ($\epsilon_\tau = 5\%$).

Chapter 6

Conclusions and Future Work

In this dissertation, we have tackled the problem of digital twin (DT) placement such that the application's data request latency targets are best accommodated. The objective is to minimize the data request response time at the application server subject to both the data age from the physical system to the application server and the data update period between the physical systems and the DT execution servers. The problem was first formulated as an integer quadratic program (IQP), which was then transformed into a semidefinite program (SDP). Given the NP-completeness of the optimization problem, exact solutions are unavailable for practical systems. Practical polynomial-time approximation algorithms were introduced for solving the placement problem that provide different trade-offs between the accommodation of the application input timing latency and the achievement of data age targets. Through several sets of simulations, it is shown that the Z-Congestion algorithm outperforms the rest in obtaining minimum constraint violation and timing latency.

In our system model and problem formulation, we have implicitly assumed that the physical devices are immobile and their distance and communication delay with their digital twins remain constant. This assumption cannot be made in dynamic environments such as streets and highways. For instance, the communication latency of a vehicle

and its digital twin can increase as the vehicle moves further away from the ES that is hosting its DT. As a potential future work for this thesis, the optimal digital twin placement problem can be investigated where the physical systems are moving objects and the digital twins have to be migrated between execution servers in order to accommodate the age of information target of the application.

Bibliography

[1] M. Grieves, *Product Lifecycle Management: Driving the Next Generation of Lean Thinking*. New York, NY, USA: McGraw-Hill Education, 2005.

[2] M. Grieves, Digital twin: Manufacturing excellence through virtual factory replication, *Digital Twin White Paper*, 2014.

[3] H. Yilong, M. Xiaoqing, S. Zhou, N. Cheng, Z. Yin, T. H. Luan, and Y. Chen, Collaboration as a service: Digital twins enabled collaborative and distributed autonomous driving, *IEEE Internet of Things Journal*, 1–1, 2022.

[4] J. Liu, Y. Shi, Z. M. Fadlullah, and N. Kato, Space-air-ground integrated network: A survey, *IEEE Communications Surveys & Tutorials*, vol. 20(4), 2714–2741, 2018.

[5] H. Cui, J. Zhang, Y. Geng, Z. Xiao, T. Sun, N. Zhang, J. Liu, Q. Wu, and X. Cao, Space-air-ground integrated network (sagin) for 6g: Requirements, architecture and challenges, *China Communications*, vol. 19(2), 90–108, 2022.

[6] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, Service entity placement for social virtual reality applications in edge computing, in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, 2018, 468–476.

[7] Y. Lu, S. Maharjan, and Y. Zhang, Adaptive edge association for wireless digital twin networks in 6g, *IEEE Internet of Things Journal*, vol. 8(22), 16 219–16 230, 2021.

[8] Q. Qi and F. Tao, Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison, *IEEE Access*, vol. 6, 3585–3593, 2018.

[9] F. Tao and M. Zhang, Digital twin shop-floor: A new shop-floor paradigm towards smart manufacturing, *IEEE Access*, vol. 5, 20 418–20 427, 2017.

[10] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, Digital twin-driven product design, manufacturing and service with big data, *International Journal of Advanced Manufacturing Technology*, vol. 94(9), 3563–3576, 2018.

[11] J. Lee, E. Lapira, B. Bagheri, and H. Kao, Recent advances and trends in predictive manufacturing systems in big data environment, *Manufacturing Letters*, vol. 1(1), 38–41, 2013, ISSN: 2213-8463.

[12] B. A. Talkhestani, N. Jazdi, W. Schloegl, and M. Weyrich, Consistency check to synchronize the digital twin of manufacturing automation based on anchor points, *Procedia CIRP*, vol. 72, 159–164, 2018, 51st CIRP Conference on Manufacturing Systems, ISSN: 2212-8271.

[13] F. Tao, Q. Qi, L. Wang, and A. Nee, Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison, *Engineering*, vol. 5(4), 653–661, 2019.

[14] H. Park, A. Easwaran, and S. Andalam, Challenges in digital twin development for cyber-physical production systems, *Cyber Physical Systems. Model-Based Design. Cham, Switzerland: Springer*, 28–48, 2018.

[15] A. K. Masudul, E. Saddik, and Abdulmotaleb, C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems, *IEEE Access*, vol. 5, 2050–2062, 2017.

[16] E. Glaessgen and D. Stargel, The digital twin paradigm for future NASA and U.S. air force vehicles, in *the 53rd Structures, Structural Dynamics, and Materials Conference*, 2012.

[17] E. J. Tuegel, A. R. Ingraffea, T. G. Eason, and S. M. Spottswood, Reengineering aircraft structural life prediction using a digital twin, *International Journal of Aerospace Engineering*, 1–14, 2011.

[18] D. Yueyue, Z. Ke, M. Sabita, and Z. Yan, Deep reinforcement learning for stochastic computation offloading in digital twin networks, *IEEE Transactions on Industrial Informatics*, vol. 17(7), 4968–4977, 2020.

[19] K. Xia, C. Sacco, M. Kirkpatrick, C. Saidy, L. Nguyen, A. Kircaliali, and R. Harik, A digital twin to train deep reinforcement learning agent for smart manufacturing plants: Environment, interfaces and intelligence, *Journal of Manufacturing Systems*, vol. 58, 210–230, 2021.

[20] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, Communication-efficient federated learning and permissioned blockchain for digital twin edge networks, *IEEE Internet of Things Journal*, vol. 8(4), 2276–2288, 2021.

[21] W. Sun, N. Xu, L. Wang, H. Zhang, and Y. Zhang, Dynamic digital twin and federated learning with incentives for air-ground networks, *IEEE Transactions on Network Science and Engineering*, vol. 9(1), 321–333, 2022.

[22] 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*, vol. 18(2), 1367–1376, 2022.

[23] D. Zhao, T. Todd, G. Karakostas, M. Vaezi, K. Noroozi, H. Wu, and X. Shen, Digital twins from a networking perspective, *IEEE Internet of Things*, 2022.

[24] T. Liu, L. Tang, W. Wang, Q. Chen, and X. Zeng, Digital-twin-assisted task offloading based on edge collaboration in the digital twin edge network, *IEEE Internet of Things Journal*, vol. 9(2), 1427–1444, 2022.

[25] W. Sun, H. Zhang, R. Wang, and Y. Zhang, Reducing offloading latency for digital twin edge networks in 6G, *IEEE Transactions on Vehicular Technology*, vol. 69(10), 12 240–12 251, 2020.

[26] H. Chunhua, F. Weicun, Z. Elan, H. Zhi, W. Fan, Q. Lianyong, and B. M. Z. Alam, Digital twin-assisted real-time traffic data prediction method for 5g-enabled internet of vehicles, *IEEE Transactions on Industrial Informatics*, vol. 18(4), 2811–2819, 2022.

[27] Z. Wang, X. Liao, X. Zhao, K. Han, P. Tiwari, M. J. Barth, and G. Wu, A digital twin paradigm: Vehicle-to-cloud based advanced driver assistance systems, in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, IEEE, 2020, 1–6.

[28] K. Zhang, J. Cao, and Y. Zhang, Adaptive digital twin and multiagent deep reinforcement learning for vehicular edge computing and networks, *IEEE Transactions on Industrial Informatics*, vol. 18(2), 1405–1413, 2022.

[29] Z. Wei, S. Wang, D. Li, F. Gui, and S. Hong, Data-driven routing: A typical application of digital twin network, in *2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI)*, 2021, 1–4.

[30] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, Low-latency federated learning and blockchain for edge association in digital twin empowered 6G networks, *IEEE transactions on Industrial Informatics*, vol. 17(7), 5098–5107, 2021.

[31] *5G Backbone*, `https://www.etsi.org/technologies/5G`.

[32] P. Tiwari, Mobility digital twin with connected vehicles and cloud computing, 2021.