DIGITAL TWIN PLACEMENT IN NETWORKS

DIGITAL TWIN PLACEMENT IN NETWORKS

By KIANA NOROOZI, M.Sc.

A Thesis Submitted to the School of Graduate Studies in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy

McMaster University

DOCTOR OF PHILOSOPHY (2024)

Hamilton, Ontario, Canada (Electrical and Computer Engineering)

TITLE:	Digital twin placement in networks
AUTHOR:	Kiana Noroozi M.Sc. (Communication Network Engineering), University of Tehran, Tehran, Iran
SUPERVISOR:	Prof. Dongmei Zhao Prof. Terence D. Todd
NUMBER OF PAGES:	xvii, 131

Lay Abstract

Digital Twins (DTs) are virtual replicas of real-world Physical Systems (PSs), such as mobile devices, vehicles, or smart cities. These digital counterparts are hosted by network servers. They mirror the state and behavior of their physical versions in real time, allowing them to interact with other devices or applications on behalf of their PSs. For a DT to effectively mirror and reflect any changes in its PS, it must consistently remain synchronized through timely updates, which consume the network resources. As a result, the placement of DTs on network servers affects the quality of the DTs. The problem becomes challenging when placing the DTs of a large number of PSs, and is further complicated when the PSs are mobile. This thesis tackles some key challenges towards optimal DT placements.

1. Optimizing Synchronization Timing and Placement: We investigate how to optimally place DTs within the network infrastructure to minimize synchronization delay. To achieve this, we develop algorithms that efficiently assign DTs to servers, balancing the need for timely updates, quick application responses, and the amount of network resources.

2. Enhancing DT Migration in Vehicular Systems: Vehicles are constantly on the move. Therefor, the PS-DT synchronization delay varies with the PS locations,

and at some point, it is better to migrate the DT to a different server. We develop algorithms that decide when to initiate the migration to minimize costs associated with the migration.

Abstract

Digital Twins (DTs) are software representations of physical systems (PSs) that interact with other entities on behalf of their real-world counterparts. To ensure accurate representation and effective interaction, DTs must remain synchronized with their PSs through timely updates—a process known as DT synchronization. This thesis addresses key challenges related to DT synchronization to optimize performance metrics, including the synchronization period and Age of Information (AoI).

In the first part, we address the challenge of optimally placing DTs on execution servers (ESs) to minimize both the data request-response delay experienced by applications and the synchronization period between PSs and their DTs, while satisfying communication and computation constraints. We formulate the DT placement problem in two ways. First, we model it as an integer quadratic program (IQP) aiming to minimize the maximum application response delay subject to maximum data age target constraints at the DTs and the application server. Due to the NP-completeness of the problem, we develop practical polynomial-time approximation algorithms that offer trade-offs between application latency and data age targets. Second, we tackle the Minimum Synchronization Period (MSP) problem by modeling it as a multi-commodity quickest flow evacuation problem, considering synchronization data and processing tasks as network flows with flow dependent edge delays. This innovative

approach allows us to use well-established techniques from flow network theory to efficiently find the quickest flow solution. An unsplittable flow rounding procedure ensures that each DT is assigned to a single ES. Simulation results demonstrate the effectiveness of our proposed algorithms in both methods, compared to optimal solutions serving as lower bounds.

In the second part, we address DT migration in vehicular systems, where maintaining acceptable AoI is challenging due to high mobility and frequent handoffs between cellular domains. We formulate the optimal initiation time for migrating a vehicle's DT as a Markov decision process, aiming to minimize the time-averaged AoI at the DT. An online optimal migration initiation algorithm is proposed using dynamic programming and optimal stopping problem. We also develop a more computationally intensive adaptive version of this algorithm, which recalculates the decision policy at each time step for improved performance. Additionally, we introduce a best-in-expectation algorithm that offers a balance between computational efficiency and AoI performance. These algorithms are compared with heuristic approaches, such as immediate migration and migration at handoff, as well as an offline algorithm providing a theoretical lower bound on the average AoI. Performance evaluations show that our proposed algorithms significantly enhance the efficiency of DT migrations while minimizing the time-averaged AoI compared to other methods.

This thesis is dedicated to the resilient young people of Iran, who in their pursuit of a better life, face immense challenges and adversities. To those who are unjustly imprisoned and deprived of basic resources, your courage and perseverance inspire my work and remind us all of the relentless pursuit of justice and equality. I also dedicate this work to my parents, whose unwavering support and belief in me have been my constant backbone. Your sacrifices have not gone unnoticed, and it is with deep gratitude that I acknowledge your unconditional love and encouragement throughout my journey.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors, Prof. Dongmei Zhao, Prof. George Karakostas, and Prof. Terence D. Todd, for their continuous support, motivation, and invaluable guidance throughout my Ph.D. study and research. Their belief in my potential and their encouragement during moments of doubt have inspired me to push beyond my limits and achieve more than I thought possible. I am thankful for the countless hours they invested in reviewing my work and providing valuable feedback that helped refine my research and successfully complete this thesis.

I extend my warm thanks to Prof. Jun Chen for his insightful suggestions. I would also like to thank the members of the examining committee for reading my thesis and providing valuable feedback.

I am deeply grateful to my parents for their unconditional love, support, and encouragement throughout my life and academic journey. Their sacrifices have provided me with the opportunities and foundation to pursue my dreams. Their wisdom and guidance have been a constant source of strength, and their faith in me has been instrumental in overcoming the challenges along the way. Without their constant support, this accomplishment would not have been possible.

A special thanks to my partner, Javad Honarvar, who has always trusted me and

stood by my side during the toughest days of my Ph.D. His constant encouragement, patience, and understanding have been invaluable. He has been my rock, offering emotional support and motivation when I needed it most. His belief in me has helped me overcome obstacles and stay focused on my goals.

I am fortunate to have many friends who have supported me and listened to me throughout this journey. I want to thank Soheila Nadalian, Dr. Armin Miremad, Dr. Maziar Shafie, Arian Aminoleslami, Dr. Mohammad Ali Maleki, Fatima Ghaderi, Mahdis Nankali, Kiana Farhadyar, Mina Ashtiani, Dr. Shabnam Ghiasvand, Dorsa Ghahramani, Dr. Danial Maleki and many more friends whom I cannot all name here. Your friendship, encouragement, and willingness to lend an ear have meant so much to me. Whether it was through thoughtful discussions, shared laughter, or simply being there when I needed someone to talk to, you all have contributed to making this journey memorable and fulfilling. I am truly grateful for your companionship and support.

Many thanks to my fellow colleagues, Hong Chen, Amir Aghaei, Mehrad Vaezi, Muhammad Heydari in the Wireless Networking Laboratory for their help and valuable discussions we had.

Finally, to everyone who has been part of this journey—teachers, colleagues, and all those who have touched my life in some way—thank you. Your contributions, big or small, have helped me reach this milestone.

Table of Contents

List of Figures

List of Tables

Definitions and Abbreviations

Abbreviations

5G	Fifth Generation
AoI	Age of Information
AS	Application Server
BS	Base Station
CPU	Central Processing Unit
DP	Dynamic Programming
DT	Digital Twin
ES	Execution Server
Gbps	Gigabits per second
ILP	Integer Linear Programming
IQP	Integer Quadratic Programming

IoT	Internet of Things
MEC	Mobile Edge Computing
MDP	Markov Decision Process
MQFP	Multi-commodity Quickest Flow Problem
MSP	Minimum Synchronization Period
PS	Physical System
QP	Quadratic Programming
SDP	Semidefinite Programming

Chapter 1

Introduction

1.1 Overview

Digital Twins (DTs) have emerged as a transformative technology in modern networked systems, offering virtual representations of physical systems (PSs) that can interact with third-party applications on behalf of their physical counterparts. By enabling applications to interface with DTs rather than directly with PSs, network traffic loads can be significantly reduced, especially when managing a large number of applications. Hosting DTs on dedicated servers allows for complex data processing and the execution of sophisticated computing models to generate diverse information required by various services and applications [78]. For these reasons, DTs have received a lot of recent attention in both academia and industry [62].

A DT is essentially a software-based implementation that mirrors a real PS, evolving alongside it throughout its operational lifetime [36, 35]. The utility of DTs has been demonstrated across numerous scenarios, notably in the Internet of Things (IoT)

and smart cities [52, 58, 29], manufacturing and industrial applications [74], and wireless communication networking [82, 24, 47]. DTs are particularly well-suited for IoT applications due to their ability to handle real-time data flow and processing. For example, a dynamic DT of vehicles and roadside units (RSUs) has been developed to capture time-varying resource demand information [93]. In vehicular networks, DTs consider the dynamic network topology to capture social features and map them into virtual spaces [68]. In industrial settings, DTs of warehouses are used to obtain real-time data and visual feedback on cargo information [16]. In the realm of cloud computing, DT-based network architectures have been proposed to replace traditional end-to-end communication models with cloud-to-end communication, allowing devices or users to be represented by their digital counterparts [92]. Integrating DTs with conventional edge networks has led to the development of digital twin edge networks, where DTs exchange information collected from their PSs, leverage the computational resources of edge servers, and generate results that enhance PS performance [90, 87]. In conventional edge computing applications, synchronizing real-time data between users and edge servers demands significant wireless resources. By mapping mobile devices to DTs within edge servers, DTs contribute to both latency reduction and reliability enhancement. Furthermore, DT-based approaches have been utilized to optimize mobile network performance by predicting future states and behaviors based on current conditions [23]. In scenarios involving unmanned aerial vehicles (UAVs), DTs facilitate optimal resource allocation by acting as intermediaries between vehicles and base stations, particularly when vehicles are outside base station coverage areas [71]. The twin models of vehicles and base stations created within UAVs help achieve efficient offloading services and resource distribution among vehicles.

A critical aspect of DTs is their need for regular updates to reflect changes in their associated PSs, a process known as DT *synchronization* [10]. This synchronization involves periodic communication between the PS and its DT to ensure that any state changes are accurately represented virtually.

The synchronized information enables the DT to provide *features* to external applications that reflect the current state and operation of the PS [78]. For instance, a DT might indicate whether an electrical circuit is functioning normally or if a mechanical subsystem requires maintenance. However, the processes of data synchronization and processing at the DT introduce latency, which impacts the *Age of Information* (AoI)—the time that has elapsed since the PS-DT update, used to generate the current DT output, was initiated [50, 89]. To maintain tight synchronization and minimize AoI, DTs are typically located close to their PSs to reduce data transmission delays [78].

As representations of their PSs, DTs can interact with applications on behalf of the physical systems. This interaction decouples communications into two parallel processes: the synchronization between the PSs and their DTs, and the information delivery between the DTs and the applications. Such decomposition introduces flexibility in network resource allocations and helps maintain a low AoI at the applications. Maintaining a low AoI is critical for applications that require timely and up-to-date information, such as those in industrial automation, real-time monitoring, smart manufacturing, [73, 69] and vehicular networks [27].

However, the placement of a DT within the network infrastructure significantly impacts both the synchronization delay between the PS and the DT and the communication delay between the DT and the application. Optimal DT placement is thus

crucial to minimize overall latency and maintain an acceptable AoI. Unlike traditional proxy server placement problems [48], DT placement must consider not only communication aspects but also the computational resources required to host the DTs and process continuous data updates from the PSs. DTs often perform complex functions beyond simple data relaying—they compress, process, and optimize data received from their PSs and provide insights based on historical information [2, 1]. These additional functionalities necessitate careful consideration of computational capacities and resource allocation during DT placement.

Dynamic PSs, especially those characterized by high mobility and real-time interactions, present significant challenges in deploying and maintaining DTS. In particular, in vehicular networks, the challenges associated with DT placement and maintenance are amplified due to the high mobility of vehicles. DTs can be created for individual vehicles [99], human drivers [41], or entire connected vehicle systems [83]. They are leveraged to make driving decisions in autonomous vehicles, enhance human driving performance, and support various applications for vehicular users. The AoI of these DTs is adversely affected by the rapid and unpredictable movements of vehicles. When a vehicle crosses a cellular domain boundary, for instance, the synchronization latency between the vehicle and its DT can increase significantly [17]. This increased latency can lead to outdated information being available to applications, potentially compromising the effectiveness and safety of vehicular systems.

To mitigate this issue, it may be desirable to migrate the DT to a more appropriate server closer to the vehicle's new location, thereby improving the AoI at the DT. This process, referred to as *digital twin migration*, introduces its own set of challenges.

Determining the optimal time to initiate DT migration is complex and requires balancing trade-offs between migration costs—such as the time and resources needed to transfer computation models and historical data—and the benefits of reduced synchronization latency and improved AoI.

Moreover, the AoI at the DT during and after migration must be carefully considered. Migration can introduce temporary increases in AoI due to the time required to transfer the DT and re-establish synchronization with the PS. This can result in periods where applications receive stale or delayed information, which is particularly problematic in safety-critical systems like vehicular networks. Despite the importance of these issues, to our knowledge, no prior work has comprehensively studied the effect of DT migration on AoI at the DT, especially in the context of highly mobile environments such as vehicular networks.

1.2 Contributions and Thesis Organization

This thesis tackles key challenges in the deployment of digital twins within wireless communication networks, specifically focusing on two critical areas: DT placement and migration. The primary objective is to minimize communication and computation delays and optimize the AoI in dynamic and resource-constrained environments. These issues are vital for ensuring the real-time performance of DT-based systems in wireless networks. By addressing these challenges, this work significantly advances the integration of DTs in complex communication network. The contributions in the thesis are summarized below.

Firstly, we formulate the DT placement problem in scenarios where multiple PSs

communicate with applications through their DTs. Recognizing the real-time requirements and the necessity for timely DT synchronization, we introduce data age targets for both applications and DTs. Each DT must be assigned to an Execution Server (ES), and this placement significantly impacts both the synchronization delay between the PS and DT and the communication delay between the DT and applications. We define the DT placement problem to minimize the maximum data request response delay experienced by applications across all PSs, while ensuring that the AoI at the DT remains within specified targets.

To address this complex problem, we show that the DT placement problem is NP-complete and formulate it as an integer quadratic program (IQP). Given the computational intractability of finding exact solutions, we develop polynomial-time approximation algorithms that provide practical DT placements. These algorithms offer trade-offs between application response times and adherence to PS data age targets. We propose various rounding methods for the fractional relaxation of the problem. Through extensive simulations, we show that our proposed methods, achieve low application interaction delays while closely meeting the specified data age targets compared to other alternatives.

Building upon the placement problem, we introduce a dynamic flow network model to accurately represent the interactions among PSs, DTs, Base Stations (BSs), and ESs, incorporating delay constraints for DT updates. This model captures both communication and computational aspects, providing a realistic framework for optimizing DT placement strategies. We formulate the DT placement challenge as a multi-commodity quickest flow problem with flow-dependent transit times. This extends traditional quickest flow problems by accommodating multiple PS-DT pairs and

variable delays dependent on flow rates. Our solution framework considers scenarios where packet routing follows either a single PS-DT path or multiple transmission paths, optimizing the placement and routing strategies together.

To tackle the complexities introduced by flow-dependent transit times, we employ a time-expanded network approach. This method transforms the dynamic problem into a static one by discretizing the planning horizon based on DT delay targets, allowing us to linearize the problem and apply efficient optimization techniques. We develop a Linear Programming (LP) model that encapsulates the complexities of the DT placement problem, including demand fulfillment, flow conservation, and capacity constraints on both transmission and computational resources. Recognizing that practical implementations require data flows to be unsplitable—meaning each DT's update data should be processed at a single ES—we propose an algorithm based on a well-known [26] approximation algorithm for unsplittable flows to adjust the splittable flows from the LP solution to unsplitable ones without increasing the maximum delay by much.

Furthermore, we focus on optimizing the initiation time for DT migration in vehicular networks to minimize the average AoI at the DT. Given the high mobility of vehicles and the resulting impact on synchronization latency and AoI, we model the DT migration initiation problem as an optimal stopping problem. This approach accounts for available statistical information of vehicle movement and aims to minimize the cost of migration within a stochastic vehicular traffic environment.

Finally, we present an online algorithm that utilizes dynamic programming and optimal stopping theory to determine the optimal migration initiation time. The algorithm meets migration time deadline constraints while achieving the minimum cost

of migration. Additionally, we introduce a best-in-expectation algorithm that offers a sub-optimal yet computationally efficient solution, which is beneficial in scenarios where the computational complexity of dynamic programming is prohibitive. We compare the performance of these algorithms with two heuristics—immediate migration and migration at handoff—and demonstrate through simulations that our proposed methods significantly reduce the average AoI compared to traditional approaches.

The rest of the thesis is organized as follows. Chapter 2 gives a brief background on digital twin technology, exploring its evolution, applications and delving into the challenges associated with DT synchronization, placement, and migration. Chapter 3 addresses the problem of DT placement for minimum application request delay with data age targets. Then, Chapter 4 introduces a dynamic flow networking approach to tackle the problem of DT placement minimizing the DT synchronization delay. In Chapter 5, we focus on optimizing the initiation timing for DT migration in vehicular networks to minimize the average AoI at the DT in a vehicular network. The thesis is concluded in Chapter 6 with suggestions for possible future work. The work in this thesis has resulted in the following publications.

- **K. Noroozi**, T.D. Todd, D. Zhao, G. Karakostas, "Digital Twin Placement Using Dynamic Flow Network Evacuation", sumbitted on IEEE International Conference on Communication, 2025.

- **K. Noroozi**, T.D. Todd, D. Zhao, George Karakostas, "Age-of-Information in Vehicular Digital Twin Migration", sumbitted on IEEE Transactions on Vehicular Technology, 2024.

- M. Vaezi, **K. Noroozi**, T. D. Todd, D. Zhao and G. Karakostas, "Digital Twin Placement for Minimum Application Request Delay With Data Age Targets,"

in IEEE Internet of Things Journal, vol. 10, no. 13, pp. 11547-11557, 1 July1, 2023, doi: 10.1109/JIOT.2023.3244424.

- M. Vaezi, **K. Noroozi**, T. D. Todd, D. Zhao and G. Karakostas, H. Wu, X. Shen, "Digital Twins From a Networking Perspective," in IEEE Internet of Things Journal, vol. 9, no. 23, pp. 23525-23544, 1 Dec.1, 2022, doi: 10.1109/JIOT.2022.3200327.

Chapter 2

Background

2.1 Introduction

This chapter provides a comprehensive background on Digital Twins (DTs) and their evolution in wireless communication networks. We begin by defining Digital Twins, outlining their key characteristics, and exploring their applications within wireless communication networks and other industries. Following this, we delve into the critical challenges associated with deploying Digital Twins in wireless networks, emphasizing the importance of optimal DT placement and how various techniques can be crucial in addressing this problem. Finally, we highlight the applications and benefits of Digital Twins in vehicular systems and discuss the specific challenges involved in DT management within this domain.

2.2 Digital Twins: Definition, Applications and Challenges

When the concept of DT was first introduced by Michael Grieves for Product Lifecycle Management (PLM) in 2003, it was intended to be a "digital equivalent to a physical product" [36]. The first practical adoption was by Tuegel et al. [77], who presented a digital framework to replicate the structural behavior of an aircraft, facilitating predictive maintenance, performance optimization, and allowing machines to interact with each other and humans [9]. As the DT concept gained traction, leading industrial companies such as Bosch, Siemens, and General Electric expanded its applications to more general physical objects [12, 7, 8]. In [52], a DT is defined as "a living model of the physical asset or system, which continually adapts to operational changes based on the collected online data and information, and can forecast the future of the corresponding physical counterpart." This definition underscores the DT's ability to mirror the PS's performance, predict potential issues, and estimate its remaining lifespan. The physical and virtual systems influence and synchronize with each other continuously, evolving together over time.

The mutual influence and synchronization between the PS and the DT have been coupled with the Internet of Things (IoT) for efficient sensing, data collection, and data processing [52, 58]. Integrating IoT technologies enables DTs to receive real-time data from sensors embedded in the PS, ensuring continuous updates and accurate representation of the PS's state.

2.2.1 Digital Twin Applications

Digital Twins have found applications across a wide range of industries due to their ability to provide real-time monitoring, predictive analytics, and optimization capabilities. Some key application areas include, manufacturing, mobile network and smart cities. In the following, we will elaborate the role of DT in each of these areas.

DTs were first introduced in the manufacturing domain to improve geometry assurance in early product design phases and to observe and study certain aspects of products without interference or removing them from service [64]. By deploying intelligent analysis and predictive performance data within the DT framework, a virtual factory can be built that is fully optimized, including factory construction, product production, life prediction, and maintenance of all industrial equipment. This achieves efficient digital management and cost reduction in manufacturing processes [85].

Additionally, DT can play a pivotal role in advancing mobile wireless networks. Advancements in mobile network infrastructure have facilitated support for strict performance requirements and introduced intelligent services at the network edge. However, making optimal decisions in such dynamic and heterogeneous networks is challenging. The deployment of DTs provides real-time monitoring of networks, supplying perception data for decision-making modules.

In mobile edge computing (MEC) networks, DTs are also utilized for making offloading decisions by creating DT networks in various configurations [70, 54, 51, 22]. For instance, in [70], the DT network includes a DT for each edge server and a DT for the entire MEC system. The DT of an edge server serves as a digital replica of the server, while the DT of the entire MEC system reflects interactions within the

system, such as mobile users' offloading decisions. By predicting future states of the MEC network, the DT network assists mobile devices in making better offloading decisions, reducing latency and system costs. Similarly, in [54], the DT network includes DTs of individual IoT devices hosted at edge servers. These DTs collect state information from devices and process it to build constantly updated DT models based on real-time data. The DTs execute computations needed for devices to make computation offloading decisions. In [51], each mobile user and edge server has its own DT, tracking the real-time status of its physical entity. The DT network maintains the real-time operating state of the network, facilitating computation offloading and resource allocation decisions. In [22], the DT network includes DTs for each device and edge server. The information maintained by the DTs constructs the network topology, monitors network conditions, and optimizes computation offloading and resource allocation decisions.

Furthermore, the concept of DTs plays a crucial role in the development and management of smart cities. With the massive amount of data gathered from IoT sensors, DTs assist in both planning and monitoring urban environments [33]. DT-integrated smart cities use information from ubiquitous multi-node IoT sensors to create a virtual city capable of monitoring, simulating, and analyzing various aspects of urban development and operations [85]. Examples of existing DT-deployed smart cities include "Virtual Singapore" [84], "Virtual Zurich" [66] and "Virtual Amaravati" [5]. In these cities, DTs are used to improve infrastructure maintenance, enhance energy efficiency, optimize transportation systems, and assist in emergency response scenarios, such as aiding firefighters during emergencies.

2.2.2 Challenges of Digital Twins

Despite the great potential of DTs, several challenges arise when developing and integrating them into practical applications. In this thesis, we tackle two of the most critical challenges:

Real-time synchronization. Maintaining tight synchronization between a PS and its DT is essential for high-quality DT performance. Achieving this requires minimizing both communication and computation delays. Any transmission delay between the DT and the PS, or among different parts of a DT, can significantly degrade the DT's quality, particularly when relying on the Internet for communication. Protocols such as Time-Sensitive Networking (TSN) [60] and Deterministic Networking (DetNet) [3] are designed to provide bounded end-to-end transmission delays. However, TSN is limited in scalability since it operates at layer-2 of Open Systems Interconnection (OSI) model, and DetNet requires the underlying network infrastructure to support its services. Moreover, ensuring reliable ultra-low end-to-end delay through these protocols necessitates complex control and management technologies [45, 76, 57]. In wireless communications, synchronization delays between the PS and its DT can be significantly affected by factors such as bandwidth allocation, network congestion, and routing inefficiencies. Limited bandwidth and variable network conditions can lead to increased transmission delays, impacting the freshness and accuracy of the DT. Efficient resource allocation is critical to mitigate these delays and ensure timely synchronization [88].

Beyond communication delays, computation times for simulation, modeling, and data processing within DTs can also be substantial. In scenarios where network and computational resources are constrained, processing large volumes of data to maintain

real-time DT operations becomes challenging. One approach to mitigate computation delays is through stage-based DT implementation, where processes are executed in phases to manage resource consumption more effectively. However, this method may not suffice for large-scale systems with high data throughput requirements, leaving the minimization of communication delays in DT synchronization as a critical challenge [79].

The dynamic nature of PSs adds another layer of complexity to the challenge of PS-DT communication. As PSs, such as vehicles or mobile devices, move between different cellular domains, they may transition from one edge server running their DT models to another. This can lead to increased synchronization delays between the PS and its DT, or even result in temporary service disconnections due to cellular hand-offs [17]. To address these issues, effective migration management techniques are essential. These techniques optimize the transfer process of DTs between edge servers, minimizing delays and ensuring seamless service continuity. By mitigating the impact of cellular hand-offs, migration management can significantly enhance both the performance and reliability of DTs in dynamic wireless environments.

Optimum resource deployment and management. Another major challenge lies in the optimal deployment and management of network resources required to implement and maintain DTs. DTs need to continuously track the real-time status of the PS and regularly update the system's features. Supporting DTs requires network resources, including communication, computing, and caching. For large and complex systems, determining the amount of each resource needed, their deployment locations, and their allocation for DTs with the desired performance is a challenging task. This requires joint management of heterogeneous resources, which becomes even

more challenging as the network scales [63]. Additionally, different applications have varying requirements in terms of feature extraction, data freshness, and similarity, leading to the need for optimizing multiple, often contradictory, objectives.

DTs can also predict network behavior by leveraging models trained on real-time and historical data. This enables networks to adapt to dynamically changing conditions proactively. However, processing the large volumes of data needed to meet real-time application requirements presents further challenges in resource management and deployment. Furthermore, the proper placement of DTs must consider the limited and heterogeneous resources at edge servers, especially when mobile devices are involved. These devices can trigger DT migrations between edge servers to maintain proximity to the PS, further complicating resource management [19].

These challenges underscore the importance of effective DT placement and DT migration strategies. In the following sections, we will discuss the critical roles of DT placement and migration management in optimizing DT performance within wireless networks, as well as introduce the associated challenges in developing and deploying these optimal strategies.

2.3 Digital Twin Placement

DT placement is a critical aspect of deploying DTs in wireless networks, as it directly influences both the synchronization delay between each DT and its corresponding PS and the communication delay between the DT and the external applications. Effective DT placement must account for various network resources, including computing, storage, and bandwidth, to ensure optimal performance and resource utilization.

The placement of DTs affects not only the latency associated with synchronization updates between the PS and the DT but also the responsiveness of interactions between the DT and applications. When DTs are placed closer to the PS, synchronization delays can be minimized, enhancing the freshness of the DT's data. Conversely, placing DTs closer to the users or applications can reduce communication delays, improving the responsiveness of services that rely on DT data. Therefore, DT placement strategies must carefully balance these factors to optimize overall system performance [95].

Moreover, providing DT services requires significant network resources. The DT placement strategy must consider the computing and storage resources required to host DTs. Efficient DT placement aims to balance these resource demands while minimizing delays in synchronization. The complexity of the DT placement problem increases when DTs interact with third-party applications. In such scenarios, DTs act on behalf of their PSs to provide information to multiple applications or when direct interaction with the PS is impractical or costly. In these cases, not only the timely DT synchronization is pivotal, but the delay in delivering information from the DT to the applications is also crucial. Therefore, DT placement strategies must consider the experienced delay between the DT and the applications to ensure timely and efficient data delivery.

2.3.1 Different Techniques for solving DT Placement

Facing these challenges, DT placement is considered as a multidimensional problem that requires careful consideration of synchronization delays, communication delays,

resource constraints, and the dynamic nature of wireless networks. Developing effective DT placement strategies is essential for realizing the full potential of digital twins in communication networks and ensuring high-quality services for applications. As a result, various methods have been proposed to address the DT placement problem, including optimization algorithms, heuristic approaches, and machine learning-based techniques. For instance, Lu et al. [55] formulated the adaptive execution server association problem, which encompasses DT placement, aiming to reduce data transmission delays and enhance user utility. Their approach adapts to dynamic network conditions, enabling efficient edge association that considers both communication costs and user requirements. Further exploring DT placement, [95] considers DT placement problems within a Mobile Edge Computing (MEC) network, accounting for the mobility of PSs and applications. They jointly consider the freshness of DT data and the service cost for applications requesting DT data. Their approach involves proposing an algorithm for the DT placement problem that minimizes the sum of the DT update costs and the total service costs for applications through efficient DT placements and resource allocation for processing user requests. Gu et al. [37] emphasize the challenges regarding computational and communication costs, underscoring the importance of intelligent DT placement strategies that balance communication and computation costs to optimize network performance.

On the other hand, approaching the DT placement problem from a flow network perspective can be promising, as this technique has shown potential in other areas like traffic routing, supply chain management, and network design, demonstrating their effectiveness in handling complex optimization problems with multiple constraints. For instance, in [44], the problem of task execution placement is mapped to computing

nodes onto a graph data structure, where edge weights and capacities encode the competing demands of data locality, fairness, and starvation-freedom. However, to the best of our knowledge, approaching the DT placement problem from a flow network perspective is a novel idea and has not been studied before.

In this thesis, we tackle the challenge of DT placement through two complementary methods aimed at minimizing the maximum data request response delay for applications interacting with multiple PSs. First, recognizing that the problem is NP-complete, and finding exact solutions is computationally infeasible, we design efficient polynomial-time approximation algorithms. These algorithms provide practical and effective DT placements under real-world constraints, ensuring that the AoI remains within specified targets.

Second, we model the PS-DT communication as a dynamic flow network, introducing the quickest flow problem to capture the time-sensitive nature of data transmission and computation. By proposing an unsplittable flow algorithm, we leverage well-established theoretical foundations to develop an optimal and scalable solution. Together, these two methods provide a robust framework for solving the DT placement problem.

2.4 Digital Twin Migration

Building DT-aided networks fundamentally relies on effectively addressing the DT placement problem. As outlined in the previous section, this involves determining the optimal association between DTs and edge servers that align with the current conditions of the PSs, minimize synchronization delays, and account for resource constraints.

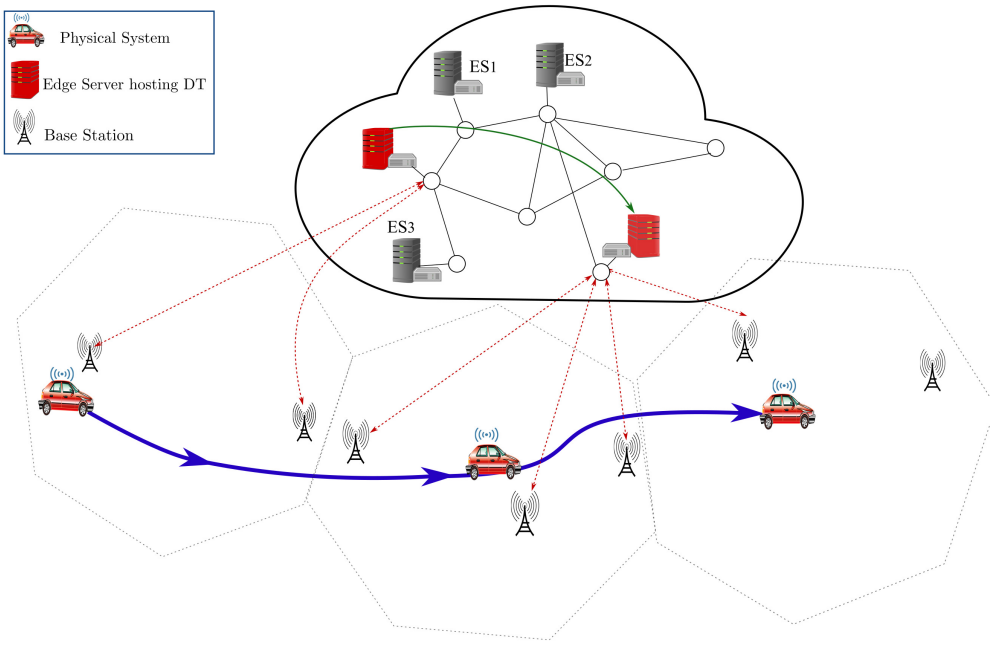


Figure 2.1: Digital Twin Migration Process

However, maintaining DTs introduces additional complexities due to the mobility of PSs. As depicted in Figure 2.1, as PSs move within the coverage area of one edge server or transition to another cellular domain managed by a different edge server, the DT must be migrated accordingly. This migration process involves transmitting the DT's historical information and reconstructing it on the target server, which is both time-consuming and resource-intensive. Consequently, the placement problem becomes intertwined with the challenge of determining the optimal timing for DT migration initiation. Effective migration strategies are essential to ensure seamless synchronization, minimize service disruptions, and optimize resource utilization in dynamic network environments. As a result, it is crucial to consider synchronization latency, resource allocation, and energy consumption in the DT migration process.

Lu et al. in [55] formulated an edge association problem aimed at reducing average

system latency and improving application utility. Their approach separates the DT placement and migration problems, addressing migration problem as the selection of an optimal server for placing DT. Similarly, [94] considers the problem of the DT migration as finding the optimal server which provide sufficient computational resources while taking into account the network's overall cost. However, enhancing the efficiency of migration and reducing synchronization delays cannot be achieved without delving deeply into the entire migration process, which typically involves not only selecting the optimal server but also determining the ideal migration initiation time, and managing data transfer.

Initiating migration promptly is essential to prevent the consumption of excessive wireless communication bandwidth and computational resources of edge servers [34], which can lead to high synchronization latency between the DT and the PSs [72]. Additionally, minimizing the added delay due to cellular hand-offs is crucial to maintaining seamless DT operations. As a result, migration initiation requires an understanding of the future of PSs, such as when vehicles cross the cellular domain boundaries. PSs are often in motion, exhibiting diverse maneuvers like acceleration, lane changes, and complex interactions with their environments. These factors, coupled with the uncertainty of sensory information and the computational burdens associated with autonomous vehicles [43], add significant complexity to accurately predicting future cellular hand-offs. This prediction involves analyzing various factors, including velocity, direction, and network conditions, to estimate when and where a PS will move out of the current edge server's coverage area.

During migration initiation, the current edge server must promptly send the up-to-date state of the PS and the DT model's parameters to the target server, e.g.

real-time sensor data, machine learning model weights, and synchronization data. Subsequently, historical data can be transferred on a priority basis [91]. Efficiently managing this data transfer process ensures that synchronization between the PS and DT is maintained, even during the transition between cellular domains. Seamless migration of DTs is essential to ensure the continuity of PS-DT communication and to avoid unnecessary power consumption, thereby enhancing both performance and reliability in dynamic environments.

The significance of DT migration strategies becomes particularly evident when considering data freshness and minimizing AoI. As highlighted in [50], placing DTs of PSs in remote cloud environments, while offering abundant computing and storage resources, may result in DT data that is not as fresh as applications expect. Additionally, measuring instantaneous AoI is impractical due to inherent delays in DT processing. Therefore, this thesis introduces an average AoI computation technique to measure overall AoI over a period of time, enabling the monitoring of the effects of optimal migration decisions.

To the best of our knowledge, the integration of both PS future prediction and the determination of optimal migration initiation timing within migration management has not been extensively explored in existing literature. This thesis addresses this gap by proposing a method for predicting PS hand-off times and developing an optimal online algorithm designed to minimize average AoI during the migration of DTs from one edge server to another. By doing so, we present a comprehensive approach that not only optimizes DT placement but also ensures efficient and timely DT migration.

Chapter 3

Digital Twin Placement with Synchronization Delay Targets

3.1 Introduction

In this chapter we consider the problem of DT placement when there are multiple PSs communicating with applications through their DTs. There are data age targets for both the applications and the DTs, which capture the real-time nature of the system and the need for DT synchronization. Each DT must be assigned to one of the available candidate ES locations. When a DT is placed, the PS periodically updates it so that its state accurately tracks that of its PS. The DT placement problem is defined so that the maximum data request response delay experienced by the application over all PSs is minimized, subject to a maximum *data age target* at the DT, i.e., the (given) data age target is an upper bound of the Age-of-Information or "freshness" of the data available at the DT [89]. The main contributions of this chapter are summarized as follows:

- A DT to Execution Server (ES) placement problem is formulated to minimize the maximum application interaction delay when accessing multiple PSs, while ensuring the timely delivery of PSs' data to their DTs and then to the application.

- The placement problem is an integer quadratic program (IQP), whose fractional relaxation can be further strengthened into a semidefinite program (SDP). The problem is shown to be NP-complete. Since an exact polynomial-time solution is not available, polynomial-time approximation algorithms are introduced to obtain DT placements, which may not be feasible for the original problem. However, the algorithms are designed to offer practical solutions that give different performance tradeoffs between application server response times and the satisfaction of PS data age targets.

- The fractional solution of the relaxed SDP is rounded to obtain the final (integral) assignments of DTs to ESs. Different rounding alternatives, i.e., Random Selection, X-Congestion, Z-Congestion, Constraint Slack SDP, and Constraint Slack QP, are proposed.

- Simulation results are provided that show Z-congestion (and, to a lesser degree, Constraint Slack SDP) achieves the best application interaction delay, while coming closer to (given) data age targets than the other alternatives.

The rest of the chapter is organized as follows. Section 3.2 summarizes the recent work related to DT placement. Section 3.3 defines the system model. The optimum DT placement problem is formulated in Section 3.4, where the original IQP is translated into an integer linear programming (ILP) and further relaxed into an SDP.

Section 3.5 presents our proposed approximation algorithms. Simulation results are shown in Section 3.6 to demonstrate the performance of the proposed algorithms.

3.2 Related Work

In this section we summarize some work related to DT placement. A more complete list of related work can be found in [78].

An algorithm is proposed in [81] for service entity placement in Virtual Reality (VR) applications. The proposed method aims at minimizing the costs of placing a service entity on an edge server. Different from DT placement in this work, the service entity placement in [81] does not contain constraints on the data update rate by the corresponding PS.

Deep-learning-based algorithms are proposed in [55] for digital twin placement and migration. Two types of DTs are considered, device DTs and service DTs. The former is a full replica of a physical device, and the latter is a lightweight DT that extracts information directly related to a specific application from multiple devices. The placement of these DTs to the servers is optimized with the objective of minimizing the average system delay.

The problem of DT placement for IoT devices in edge IoT networks is studied in[18]. The objective is to minimize the total communication latencies between the IoT devices and their corresponding DTs. One assumption behind this formulation is sufficient server capacity so that the data processing time at the DTs can be neglected.

Different from the above work, our work addresses the DT placement issue by considering the age-of-information at both the DTs and the application. We employ a

time-sharing resource allocation model for computational resources at the servers, although this results in an integer quadratic problem that is proven to be NP-complete. Our objective is to minimize the maximum delay experienced by an application that is accessing information from multiple DTs. There is prior work such as [75, 81] where digital twins are optimized to serve particular applications. However, to the best of our knowledge, ours is the first work that places a maximum data age target constraint on the data provided by the DTs to the application.

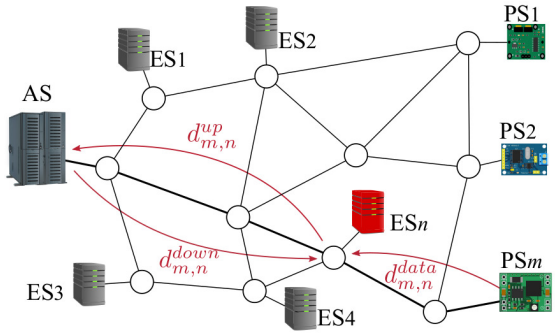


Figure 3.1: Digital Twin Placement Model

3.3 System Model

Figure 3.1 shows an application server (AS) that requires input from multiple PSs that are connected to the same network. Each PS has an associated DT that it regularly communicates with so that the latter can interact with the application on its behalf. Each DT is hosted at one of a set of execution servers, ESs, which are also shown in the figure. A practical example of this is where the PSs include sensors located in a manufacturing plant and provide their inputs at regular intervals so that the application can monitor the performance of the system [2][1].

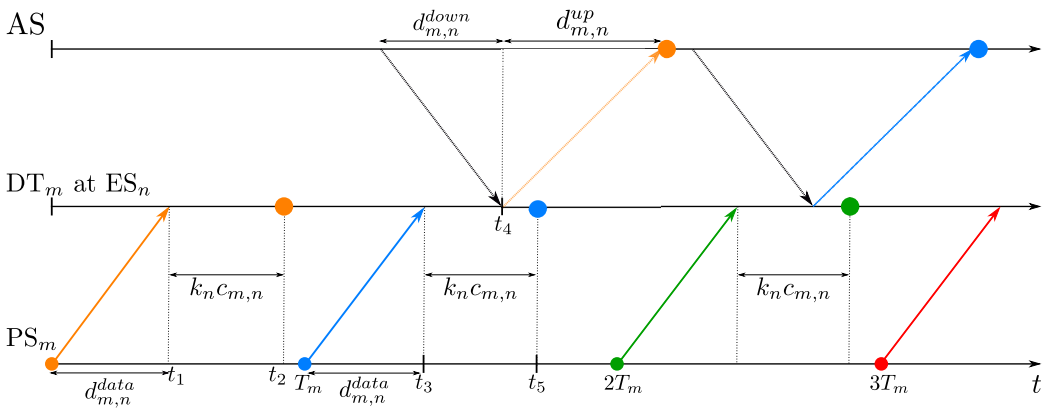


Figure 3.2: DT Update and AS Request Timing

The main objective of this chapter is to determine the placement of the DTs at the ESs so that the maximum communication delay experienced by the application is minimized, and at the same time, the data delivered best adheres to a desired data age target. When placed at an ES, a DT incurs both communication and computational delays as its PS data is periodically refreshed. Let $\mathcal{M} = \{1, 2, ..., M\}$ be a set of M PSs and $\mathcal{N} = \{1, 2, ...N\}$ be a set of N ESs. Each PS has an associated DT, i.e., DT_m for PS_m that must be placed at one of the ESs, e.g., ES_n for $n \in \mathcal{N}$.

Figure 3.2 shows a timeline example of PS_m, DT_m (at ES_n), and the AS. Each PS periodically updates its DT with new data as shown in the bottom two timelines. At $t = 0$, the first update (shown by the purple dot) is transferred from PS_m to DT_m (at ES_n) in time $d^{data}_{m,n}$ and is received by the ES server at time t_1. At DT_m, this update requires $c_{m,n}$ seconds of computation from ES_n before the update is completed. If k DTs have been assigned to ES_n then, as in the colocation constraints in [81], the processing will be completed after a delay no longer than $k \cdot c_{m,n}$ seconds, at time t_2 in the figure. This updating procedure recurs periodically for each PS, i.e., for PS_m it repeats every T_m seconds. At $t = T_m$ and $t = 2T_m$, the second and third PS_m

updates (at the blue and green dots, respectively) are shown in Figure 3.2.

The top timeline (AS) in Figure 3.2 shows the application generating a request for input from DT_m. This request arrives at the DT $d_{m,n}^{down}$ seconds later, and the requested data are returned to the AS after a further delay of $d_{m,n}^{up}$ seconds. These time durations are also shown by the arrows in Figure 3.1. A DT always responds to application requests immediately with the most recently updated version of the requested data. In Figure 3.2, the first application request arrives at the DT at $t = t_4$, and therefore, the DT passes the results based on processing the first update to the application server since the second update has not completed.

One of the data age requirements is that the PS_m data available at DT_m must be synchronized every T_m seconds, which implies that the time needed for m's data to be transmitted to and processed by DT_m should not be more than T_m seconds. That is,

$$d_{m,n}^{data} + k_n \cdot c_{m,n} \leq T_m, \tag{3.3.1}$$

where k_n is the total number of DTs hosted by ES_n.

The application may query multiple DTs in order to obtain the information that it needs. It has its own data age target, denoted by A^*, which is the desired maximum age target over all the queried PSs, i.e., if the target is satisfied, then when data arrives from all the queried DTs, the time since the data was generated at all the PSs will be at most A^* seconds. We also assume that there are known upper bounds on data transfer latencies between ES_n and the application server and each PS [61]. These bounds can be guaranteed, e.g., by resource allocation via contractual terms with the network provider, and by power control when the PS has a wireless connection to the

network [86][28].

Figure 3.2 shows the worst case for the age of the PS_m data delivered to the application when the data request from the AS arrives just before the next PS_m-to-DT_m data update cycle has completed, i.e., $t_4 < t_5$ and $t_5 - t_4 \approx 0$. This coincides with the time $T_m + d^{data}_{m,n} + k_n \cdot c_{m,n}$ in the figure, when the processing of data generated at time T_m (the blue one) is almost (but not quite) ready at DT_m. As a result, the application receives the data generated at time $t = 0$ (the purple one in the figure) $T_m + d^{data}_{m,n} + k_n \cdot c_{m,n} + d^{up}_{m,n}$ seconds later. According to the discussion above, this worst-case time cannot be larger than A^* if the application data age target is to be achieved, i.e.,

$$T_m + d^{data}_{m,n} + k_n \cdot c_{m,n} + d^{up}_{m,n} \leq A^*. \tag{3.3.2}$$

As in [61], we assume that network latencies are known or can be estimated. Since the loading imposed on the network is assumed to be small, we also assume that they are independent of the DT placement, i.e., the aggregate loading of the PSs communicating with the ESs is assumed to be far less than the actual capacity of the underlying links connecting the ESs, as in [81].

Note that for a given set of upper bounds in (3.3.1) and (3.3.2), there may not exist a feasible placement of DTs to ESs, i.e., at least one of these constraints will be violated no matter where the DTs are placed. We show below that detecting the infeasibility of the given input (and, therefore, calculating an optimal placement, if such a placement exists) is NP-complete (see Theorem 3.4.1). Therefore, our proposed algorithms may violate a number of the constraints.The algorithms, however, work towards incurring the smallest possible violation of (3.3.1) and (3.3.2), and, therefore,

they work even if the original input is infeasible. Nevertheless, in order to study their performance, we compare them only on feasible instances in Section 3.6, since we do not want to combine their own potential inefficiencies with inefficiencies inherent in the input itself.

3.4 Problem Formulation

We define binary decision variables $X_{m,n} \in \{0,1\}$ for $m = 1,2,\ldots,M$ and $n = 1,2,\ldots,N$ with $X_{m,n} = 1$ if DT_m is placed on ES n and $X_{m,n} = 0$ otherwise. Then conditions (3.3.1) and (3.3.2) become

$$d_{m,n}^{data} + c_{m,n} X_{m,n} \sum_{k=1}^{M} X_{k,n} \leq T_m, \tag{3.4.1}$$

and

$$X_{m,n} \left(T_m + d_{m,n}^{data} + c_{m,n} \sum_{k=1}^{M} X_{k,n} + d_{m,n}^{up} \right) \leq A^*, \tag{3.4.2}$$

respectively.

By addressing the problem of placing the DTs of M PSs on N ESs, our objective is that the maximum data request response delay experienced by the application over all PSs is minimized, while its data age requirement, and the refreshing rate requirement of all DTs are satisfied. This is formulated as the following Integer Quadratic Program (IQP):

$$\min_{X,\tau} \ \tau \text{ s.t.} \tag{IQP}$$

$$\sum_{n=1}^{N}(d_{m,n}^{down}+d_{m,n}^{up})X_{m,n}\leq\tau,\ \forall m\in\mathcal{M} \tag{3.4.3}$$

$$d_{m,n}^{data}+c_{m,n}X_{m,n}\sum_{k=1}^{M}X_{k,n}\leq T_m,\ \forall m\in\mathcal{M},n\in\mathcal{N} \tag{3.4.4}$$

$$X_{m,n}(T_m+c_{m,n}\sum_{k=1}^{M}X_{k,n}+d_{m,n}^{data}+d_{m,n}^{up})\leq A^*,\forall m,n \tag{3.4.5}$$

$$\sum_{n=1}^{N}X_{m,n}=1,\ \forall m\in\mathcal{M} \tag{3.4.6}$$

$$X_{m,n}\in\{0,1\},\ \forall m\in\mathcal{M},n\in\mathcal{N} \tag{3.4.7}$$

$$\tau\geq 0, \tag{3.4.8}$$

where the LHS of constraint (3.4.3) is the delay experienced by the application when requesting the PS m data, τ is the maximum of such delays over all m, and we seek to minimize τ. Constraint (3.4.6) ensures that a DT is assigned to one ES only, for all M DTs.

Constraint (3.4.4) can be rewritten as

$$X_{m,n}\sum_{k=1}^{M}X_{k,n}\leq\frac{T_m-d_{m,n}^{data}}{c_{m,n}}, \tag{3.4.9}$$

and constraint (3.4.5) as

$$X_{m,n}\sum_{k=1}^{M}X_{k,n}\leq\frac{A^*-X_{m,n}(T_m+d_{m,n}^{data}+d_{m,n}^{up})}{c_{m,n}}, \tag{3.4.10}$$

for all m,n. Now, both constraint (3.4.9) and (3.4.10) share the same LHS and can be merged into one constraint

$$X_{m,n}\sum_{k=1}^{M}X_{k,n}\leq u_{m,n} \tag{3.4.11}$$

where

$$u_{m,n} = \left\lfloor \min \left\{ \frac{T_m - d_{m,n}^{data}}{c_{m,n}}, \frac{A^* - (T_m + d_{m,n}^{data} + d_{m,n}^{up})}{c_{m,n}} \right\} \right\rfloor \tag{3.4.12}$$

Hence, values for $\mathbf{X}$ satisfy (3.4.4), (3.4.5) iff these values satisfy (3.4.11). The new formulation can be written as:

$$\min_{X,\tau} \ \tau \text{ s.t.} \tag{IQP'}$$

$$\sum_{n=1}^{N} (d_{m,n}^{down} + d_{m,n}^{up}) X_{m,n} \leq \tau, \ \forall m \in \mathcal{M} \tag{3.4.13}$$

$$X_{m,n} \sum_{k=1}^{M} X_{k,n} \leq u_{m,n}, \ \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{3.4.14}$$

$$\sum_{n=1}^{N} X_{m,n} = 1, \ \forall m \in \mathcal{M} \tag{3.4.15}$$

$$X_{m,n} \in \{0,1\}, \ \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{3.4.16}$$

$$\tau \geq 0 \tag{3.4.17}$$

For a specific τ, all variables $X_{m,n}$ in (3.4.13) with coefficients $d_{m,n}^{down} + d_{m,n}^{up} > \tau$ are forced to be 0 because of (3.4.15). Therefore, we can perform a binary search in range $[0, \max_{m,n}\{d_{m,n}^{down} + d_{m,n}^{up}\}]$ for the minimum feasible τ. Every value of τ, in effect, defines a bipartite graph $G = (A, B, E)$ with the nodes in A corresponding to DTs, the nodes in B corresponding to ESs, and edges $(m, n) \in E$ only when $d_{m,n}^{down} + d_{m,n}^{up} \leq \tau$. Therefore, for any value of τ, we can simplify problem (IQP') to the question of feasibility of constraints (3.4.14)-(3.4.16) on a bipartite graph (where $X_{m,n} := 0$ whenever $(m, n) \notin E$). We adopt this approach when we study the τ value and the violation of feasibility of (3.4.14) in Section 3.6. Problem (3.4.14)-(3.4.16) on a bipartite graph is an NP-complete problem, as shown by the following theorem.

Theorem 3.4.1. *Deciding the feasibility of* (3.4.14)-(3.4.16) *on a bipartite graph is an NP-complete problem.*

Proof. The problem is clearly in NP, since, given a $\{0,1\}$-assignment for variables $X_{m,n}$, checking the feasibility of constraints (3.4.14), (3.4.15) can be done in polynomial time.

We reduce SATISFIABILITY (i.e., given a CNF (Conjunctive Normal Form) formula, asking whether there is a satisfying truth assignment for its variables) to our problem. 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, i.e., one node for each variable or clause. On the right-side B there are $2n$ nodes, i.e., a pair of nodes for every pair of literals $x_i, \bar{x_i}$ corresponding to the 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 (3.4.11) for $m = i, n = x_i$, i.e., only variables $X_{k,\bar{x_i}}$ will be allowed to take value 1, for clauses k that 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 (3.4.14)-(3.4.16). If the formula is satisfiable, then set $X_{i,x_i} = 1, X_{i,\bar{x_i}} = 0$ if $x_i = 0$, or $X_{i,x_i} = 0, X_{i,\bar{x_i}} = 1$ if $x_i = 1$. Also, each clause l must contain a literal x_i or $\bar{x_i}$ that is set to 1, and so 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 (3.4.14)-(3.4.16). Conversely, if there is a value assignment to variables $X_{m,n}$ that satisfies (3.4.14)-(3.4.16), 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. □

The problem is clearly in NP, since, given a $\{0,1\}$-assignment for variables $X_{m,n}$, checking the feasibility of constraints (3.4.14), (3.4.15) can be done in polynomial time. As a result of Theorem 3.4.1, we do not expect that there is a polynomial-time algorithm that exactly solves (IQP').

We can linearize problem (IQP') as follows. We use a binary variable $Z^n_{k,m} \in \{0,1\}$ to replace the product $X_{k,n}X_{m,n}$ and add the following valid constraints:

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

The new problem formulation is an integer linear programming (ILP) given as follows:

$$\min_{X,Z,\tau} \ \tau \text{ s.t.} \tag{ILP}$$

$$\textstyle\sum_n (d^{down}_{m,n} + d^{up}_{m,n}) X_{m,n} \leq \tau, \ \forall m \in \mathcal{M} \tag{3.4.18}$$

$$\textstyle\sum_k Z^n_{k,m} \leq u_{m,n}, \quad \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{3.4.19}$$

$$\textstyle\sum_n X_{m,n} = 1, \ \forall m \in \mathcal{M} \tag{3.4.20}$$

$$Z^n_{k,m} = Z^n_{m,k},\ \forall k, m \in \mathcal{M}, n \in \mathcal{N} \tag{3.4.21}$$

$$Z^n_{m,m} = X_{m,n},\ \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{3.4.22}$$

$$Z^n_{k,m} \leq X_{m,n},\ \forall k, m \in \mathcal{M}, n \in \mathcal{N} \tag{3.4.23}$$

$$X_{m,n} + X_{k,n} - Z^n_{k,m} \leq 1,\ \forall k, m \in \mathcal{M}, n \in \mathcal{N} \tag{3.4.24}$$

$$X_{m,n}, Z^n_{k,m} \in \{0, 1\},\ \forall k, m \in \mathcal{M}, n \in \mathcal{N} \tag{3.4.25}$$

$$\tau \geq 0 \tag{3.4.26}$$

With these added constraints, any integral feasible solution sets $Z^n_{k,m} = X_{k,n}X_{m,n},\ \forall k, m, n$. Note that for (3.4.22), $Z^n_{m,m} = X^2_{m,n} = X_{m,n}$.

We can strengthen problem (ILP) by observing that when $X_{m,n}$'s have integral values, the matrices $\mathbf{Z}^n$ are positive semi-definite (PSD) for all n, since $\mathbf{Z}^n = \mathbf{X}^n\mathbf{X}^{nT}$, where $\mathbf{X}^n$ is the n-th column of matrix $\mathbf{X} = [X_{m,n}]$. Therefore we can add the following constraint into problem (ILP)

$$\mathbf{Z}^n \in \mathcal{PSD},\ \forall n \in \mathcal{N}. \tag{3.4.27}$$

If we relax (3.4.25) to $X_{m,n}, Z^n_{m,k} \geq 0,\ \forall k, m \in \mathcal{M}, n \in \mathcal{N}$, (ILP) becomes the following problem

$$\min_{X,Z,\tau}\ \tau \text{ s.t.} \tag{SDP-relaxed}$$

$$(3.4.18) - (3.4.24), (3.4.26), (3.4.27)$$

$$X_{mn}, Z^n_{k,m} \geq 0, \quad \forall k, m \in \mathcal{M}, n \in \mathcal{N}. \tag{3.4.28}$$

This relaxed problem will be referred to as the *SDP-relaxed problem*. It is a convex

SDP program and can be solved in polynomial time by any SDP solver. Note that constraints $X_{m,n} \leq 1$ and $Z^n_{m,k} \leq 1$ are implied by (3.4.20), (3.4.23).

As noted in Section 3.3, it may be the case that for a given input, (SDP-relaxed) is infeasible. This implies that the integer problem (IQP'), as well as the original problem (IQP) are also infeasible. In the case when (IQP') is infeasible, but (SDP-relaxed) is feasible, our algorithms are not affected, since they round the fractional solution of (SDP-relaxed). If (SDP-relaxed) is infeasible, (3.4.19) is modified by "inflating" the upper bounds with $\lambda u_{m,n}$, where $\lambda > 1$. Let $\lambda_{\max} = M/(\min_{\forall\ m,n} u_{m,n})$. Setting $\lambda = \lambda_{\max}$ allows the modified (3.4.19) to be always satisfied. By running a binary search on the interval $[1, \lambda_{\max}]$, the smallest λ can be found such that (SDP-relaxed) becomes feasible. Our algorithms will work with the fractional solution corresponding to these new (inflated) upper bounds. Therefore, for the rest of the chapter, we will assume that (SDP-relaxed) is feasible.

3.5 Approximation Algorithms

The problem formulation of Section 3.4 treats constraint (3.4.11) as a *hard* constraint that would be satisfied if an optimal solution to (IQP') was available. For practical systems this is not possible since even finding a feasible (not necessarily optimal) integral solution to problem (IQP') is NP-complete (Theorem 3.4.1). Therefore, in this section we introduce polynomial-time approximation algorithms for solving the DT placement problem. The algorithms we propose will return solutions that are approximate in terms of the objective τ, and also will violate the merged DT refresh and application data age constraints (3.4.19).

In Section 3.6 we include comparisons of the proposed algorithms in cases where

there is an integral optimum, i.e., the original problem instance is feasible for (3.4.3)-(3.4.8).

Rounding algorithm: We start by describing how to round a fractional solution of the (SDP-relaxed) 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 detect its infeasibility (Algorithms 2 & 7).

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

While $\mathcal{SM} \neq \emptyset$, the algorithm picks a DT m from $\mathcal{SM}$ and a pair of ESs n_1, n_2 with $0 < X_{m,n_1}, X_{m,n_2} < 1$ (line 8). Different selection criteria give different rounding algorithms, and will be described in detail below.

We set $X_{m,n_1} := X_{m,n_1} + X_{m,n_2}, X_{m,n_2} := 0$, resulting in at least one more rounded variable of $\mathbf{X}$ (lines 9-13). Note that this update may violate the constraints. Therefore, the algorithm goes through a series of updates to make the problem to be feasible again. This includes increasing T_m's and A^* to satisfy constraint (3.4.19) (lines 16-27) and adjusting the $Z^n_{k,m}$ values based on the updated $X_{m,n}$'s to satisfy constraints (3.4.23) and (3.4.24) (lines 29-36). Note that fixing constraints (3.4.24) (line 34) after constraints (3.4.23) (line 31) ensures that the latter will still be satisfied.

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

Let

$$\Delta u = \max_{m,n} \frac{u_{m,n} - u^*_{m,n}}{u^*_{m,n}} \tag{3.5.1}$$

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

As already mentioned, the choice of m, n_1, n_2 in Algorithm 1 (line 8) gives rise to different rounding algorithms described below.

- **Random Selection:** Pick uniformly at random a DT $m \in \mathcal{SM}$ and two ESs $n_1, n_2 \in \mathcal{N}$ such that $0 < X_{m,n_1}, X_{m,n_2} < 1$.

- **X-Congestion:** $\sum_m X_{m,n}$ is used as a measure of the *congestion* of ES n. Let $n_2 = \arg\max_n \sum_{m=1}^{M} X_{m,n}$, i.e., n_2 is the most congested ES, and $m = \arg\max_k X_{k,n_2}$. Set $n_1 = \arg\min_n X_{m,n}$ (we break ties arbitrarily).

- **Z-Congestion:** $\sum_{k,m} Z^n_{k,m}$ is used as a measure of the congestion of ES n. Let $n_2 = \arg\max_n \sum_{k,m} Z^n_{k,m}$. Then m, n_1 are picked as in **X-Congestion**.

- **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 (3.4.19) is minimum. Note that if any of these constraints is tight, the minimum slack is 0. Set $n_1 = \arg\max_n u_{m,n}$, i.e., n_1 is the ES that has the maximum slack among all the constraints (3.4.19) for m. 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:** Same as **Constraint Slack SDP**, only now we consider the slack of constraints (3.4.11).

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 $O(M^2N^2)$.

Approximation algorithms: We use Algorithm 1 as a subroutine to develop algorithms that work towards restricting deviation from fractional u or τ (Algorithms 2 and 7, respectively).

Algorithm 2 Finding solution with sub-ϵ_u violation

Input: $D_{sorted} = \{\hat{\tau}_1, \hat{\tau}_2, \ldots, \hat{\tau}_{M,N}\}, \epsilon_u$

1: $\hat{\tau}_f$ = min in D_{sorted} s.t. (SDP-relaxed) is feasible (binary search in D_{sorted})

2: $\hat{\tau}_s$ = min in $\{\hat{\tau}_f, \hat{\tau}_{f+1}, \ldots, \hat{\tau}_{M,N}\}$ s.t. {Binary search}

- X_f, Z_f = solution of (SDP-relaxed) problem with $\tau = \hat{\tau}_s$
- $X, Z, \Delta u$ = Algorithm 1(X_f, Z_f)
- $\Delta u \leq \epsilon_u$

3: **if** no $\hat{\tau}_s$ **then**

4: return INFEASIBLE

5: **else**

6: return $\mathbf{X}$

7: **end if**

Algorithm 3 Finding solution with sub-ϵ_τ violation

Input: $D_{sorted} = \{\hat{\tau}_1, \hat{\tau}_2, \ldots, \hat{\tau}_{M,N}\}, \epsilon_\tau$

1: τ_{opt}^{SP} = fractional optimum of (SDP-relaxed)
2: Find max s such that $\hat{\tau}_s \leq \tau_{opt}^{SP}(1 + \epsilon_\tau)$
3: X_f, Z_f = solution of (SDP-relaxed) problem with $\tau = \hat{\tau}_s$
4: $X, Z, \Delta u$ = Algorithm 1(X_f, Z_f)
5: **if** no $\hat{\tau}_s$ **then**
6: return INFEASIBLE
7: **else**
8: return **X**
9: **end if**

Let τ_{opt}^{QP} be the optimum τ of (IQP'). 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. More specifically, define $D_{sorted} = \{\hat{\tau}_1, \hat{\tau}_2, \ldots, \hat{\tau}_{M,N}\}$ to be 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 $|D_{sorted}| < MN$, but for clarity of the presentation we will assume that all these values are distinct). By fixing $\tau := \hat{\tau}_s \in D_{sorted}$, (SDP-relaxed) becomes a *feasibility* problem as follows: In order to satisfy (3.4.18), $X_{m,n} = 0$ for all m and n with $d_{m,n}^{up} + d_{m,n}^{down} > \hat{\tau}_s$ must be true. Therefore, the following constraints are added:

$$X_{m,n} = 0, \forall m, n : d_{m,n}^{up} + d_{m,n}^{down} \in \{\hat{\tau}_{s+1}, \ldots, \hat{\tau}_{m,n}, \} \tag{3.5.2}$$

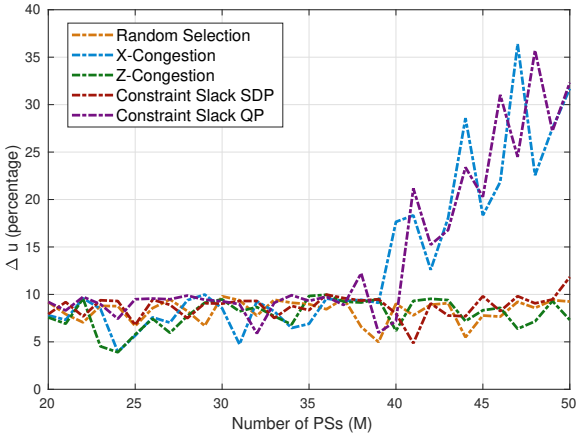
and are considered together with constraints (3.4.19)-(3.4.25) and (3.4.27). The relaxation of this feasibility problem is the new (SDP-relaxed) problem, and a (fractional) feasible solution can be obtained in polynomial time. If the problem is infeasible, then we proceed with a different (smaller) value of τ. Recall that we have assumed that the original (SDP-relaxed) is feasible, so there is at least one value of τ for which we will obtain a fractional feasible solution.

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

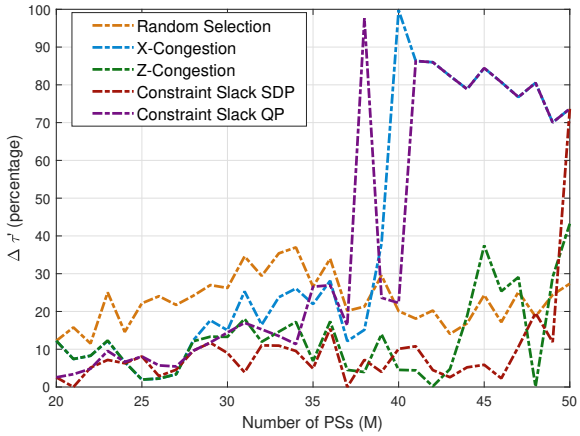
In Algorithm 2 we were aiming to find a solution with sub-ϵ_u constraint violation. Similarly, an ϵ_τ bound can be applied to the objective τ as follows: Let

$$\Delta\tau(x) = \frac{x - \tau_{opt}^{SP}}{\tau_{opt}^{SP}}, \tag{3.5.3}$$

where τ_{opt}^{SP} is the (fractional) optimum of the (SDP-relaxed). After calculating τ_{opt}^{SP}, the algorithm chooses the largest $\hat{\tau}_s$ from the set D_{sorted} with $\Delta\tau(\hat{\tau}_s) \leq \epsilon_\tau$. The (SDP-relaxed) 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 terminates with infeasibility.



(a) Δu versus M

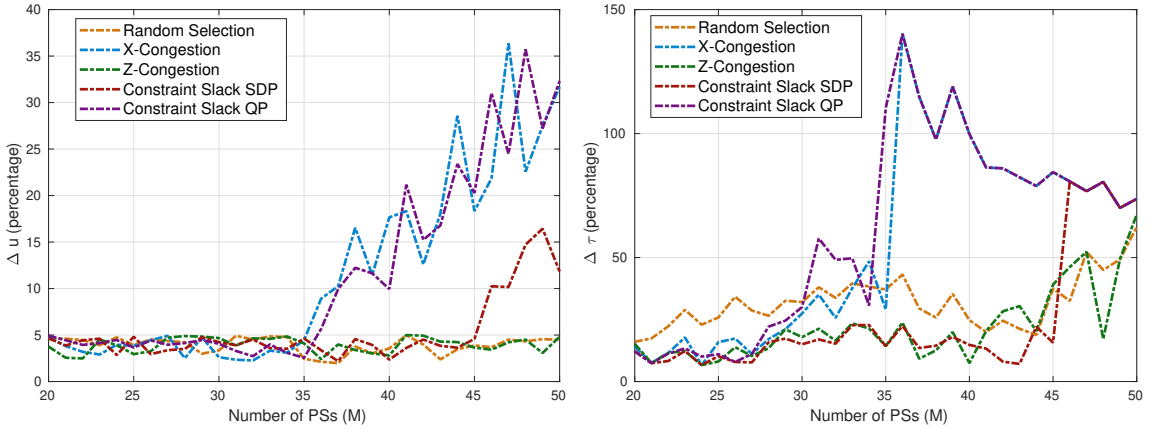


(b) $\Delta\tau'$ versus M

Figure 3.3: Performance of Algorithm 2, $\epsilon_u = 10\%$.

3.6 Simulation Results

In this section we evaluate the performance of our proposed algorithms with via computer simulation. The algorithms are implemented using the five selection methods described in Section 3.5. 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. Mosek optimization toolbox [59] was used in



(a) Δu versus M (b) $\Delta\tau'$ versus M

Figure 3.4: Performance of Algorithm 2, $\epsilon_u = 5\%$.

Matlab for solving the optimization problems. In the presented figures, each point represents an average of 50 simulation runs. Since constraint (3.4.14) 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 grouped in three groups: one that is nearest to the PSs and furthest from the AS (group 1), one that is in medium range from the PSs and AS (group 2), and one that is nearest to the AS and furthest from the PSs (group 3). The delays are defined accordingly:

- The PS-ES delays d^{data} will be uniformly distributed in ranges $[5ms, 10ms]$, $[12.5ms, 17.5ms]$, $[20ms, 25ms]$ for the three groups, respectively.
- The AS-ES delays d^{down} will be uniformly distributed in ranges $[0.7ms, 1.1ms]$, $[0.4ms, 0.8ms]$, $[0.1ms, 0.5ms]$ for the three groups, respectively.
- The ES-AS delays d^{up} will be uniformly distributed in ranges $[16ms, 20ms]$, $[10ms, 14ms]$, $[4ms, 8ms]$ for the three groups, respectively.

These delay ranges are chosen by considering the fact that the 5G network backbone is capable of supporting bit rates ranging from 5G bits/s up to 10G bits/s at stable and uncongested network conditions [6] and some link processing overhead.

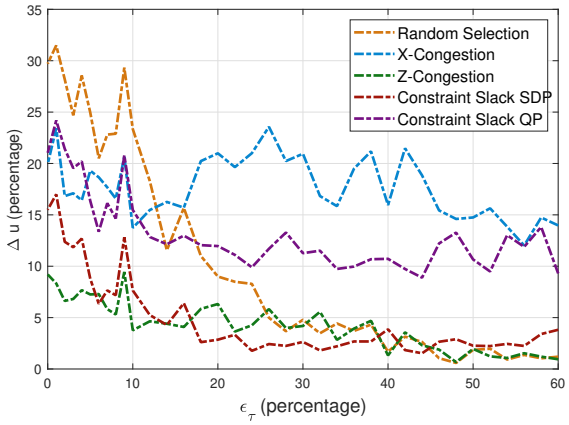


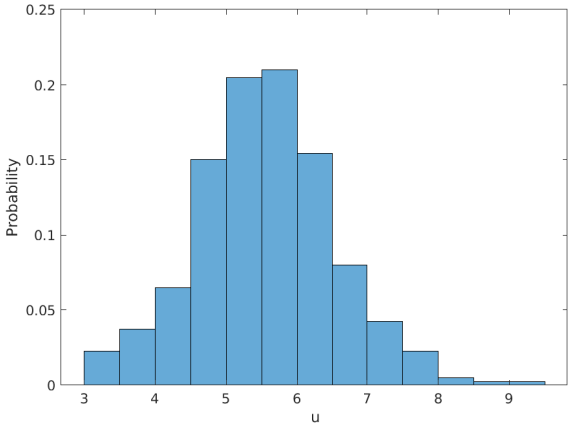
Figure 3.5: Performance of Algorithm 7, Δu versus ϵ_τ, $M = 40$.

3.6.1 Simulation set 1

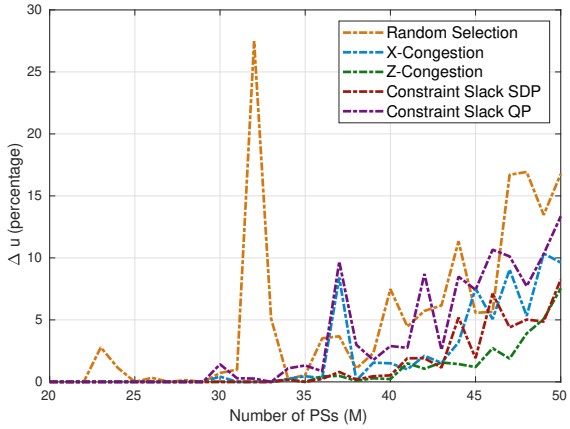
For the first set of simulations, we assigned 2 ESs to group 1, 4 ESs to group 2, and 2 ESs to group 3. The size of the data sent by a PS to its DT in every data update cycle is 25MB [75], the data size sent from a DT to the application server is 20MB [75], and the size of the request data from the application server to the DTs is 1MB [75]. In addition, T_m is uniformly distributed between 60ms and 80ms for each PS, and A^* is 200ms.

First, Algorithm 2 was run by varying the number of PSs. The results are plotted in figures 3.3a and 3.3b for $\epsilon_u = 10\%$ and figures 3.4a and 3.4b for $\epsilon_u = 5\%$. Figure 3.3a shows that all the methods can keep the Δu value below ϵ_u for a certain range of M values. However, compared to Constraint Slack QP and X-Congestion, Z-Congestion, Constraint Slack SDP, and Random Selection maintain $\Delta u < \epsilon_u$ for

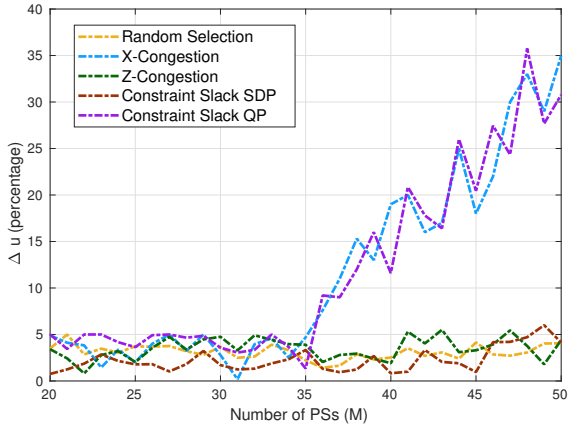
larger M values. The same trend is also seen in Figure 3.4a, although as ϵ_u becomes smaller, the range of M values for which $\Delta u < \epsilon_u$ is smaller for all methods, except for Z-congestion and Random Selection, which keep $\Delta < \epsilon_u$ for the entire range of M in the simulations.



(a) Probability distribution of u



(b) Δu versus M, $\epsilon_\tau = 5\%$



(c) Δu versus M, $\epsilon_u = 5\%$

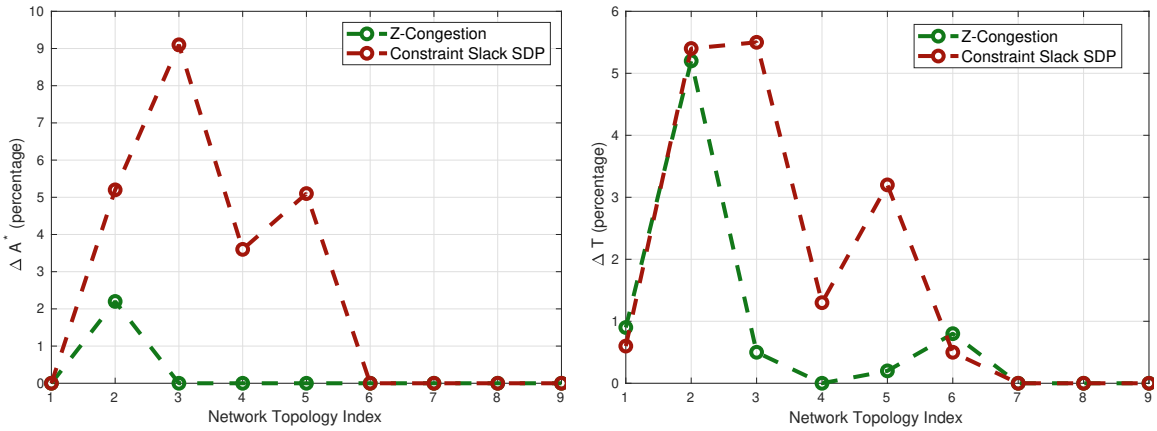
Figure 3.6: Constraint violation of algorithms over a non-uniform distribution of u.

Figures 3.3b and 3.4b show $\Delta\tau'$ values as the number of PSs change, where $\Delta\tau'$ is the relative difference between the resulted τ value after running Algorithm 2 and

the optimum integer solution. More specifically, $\Delta\tau'$ is defined as

$$\Delta\tau' = \frac{\hat{\tau} - \tau_{opt}}{\tau_{opt}}, \tag{3.6.1}$$

where $\hat{\tau}$ is the objective value achieved by Algorithm 2, and τ_{opt} is obtained by solving (SDP-relaxed) in Section 3.4. Both figures 3.3b and 3.4b show that when M is too large for the algorithm to keep $\Delta u < \epsilon_u$, the corresponding $\Delta\tau'$ is also large. However, the Z-Congestion and Constraint Slack SDP achieve much smaller $\Delta\tau'$ than the other methods. A comparison between Figures 3.3b and 3.4b shows that a larger ϵ_u value helps reduce $\Delta\tau'$, as expected.



(a) (b)

Figure 3.7: Performance of Constraint Slack SDP and Z-Congestion, $\epsilon_\tau = 5\%$ and $M = 30$.

Next, Algorithm 7 was run with different ϵ_τ values and for each selection method. The corresponding maximum constraint violation Δu is given in Figure 3.5 as a percentage over its original u bound. In general, when ϵ_τ increases, the algorithm allows a larger $\Delta\tau$ value, which helps reduce the Δu value. This tradeoff between Δu and $\Delta\tau$ is well reflected in the Z-Congestion, the Constraint Slack SDP, and the

Random Selection, but not obviously reflected in the X-Congestion and Constraint Slack QP. As can be observed from Figure 3.5, Z-Congestion and Constraint Slack SDP achieve consistently smaller Δu values than Random Selection, Constraint Slack QP and X-Congestion for the entire range of ϵ_τ values.

Based on the above results, Constraint Slack SDP and Z-Congestion offer better overall performance than the other selection methods. This is not surprising, since these two methods use the constraints of (SDP-relaxed) as a guide for rounding its own fractional solution, while Constraint Slack QP and X-Congestion round the (SDP-relaxed) fractional solution using the constraints of (IQP') as their rounding criterion.

3.6.2 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 Section 3.6.1. Instead of specifying the values of T_m's, A^*, the $u_{m,n}$ values are randomly generated from the distribution of Figure 3.6a, which emulates a normal-like distribution. The results of running Algorithm 2 and 7 on the generated instances are plotted in Figures 3.6b and 3.6c. Again, Z-Congestion gives consistently lower Δu values, when compared to the other rounding methods.

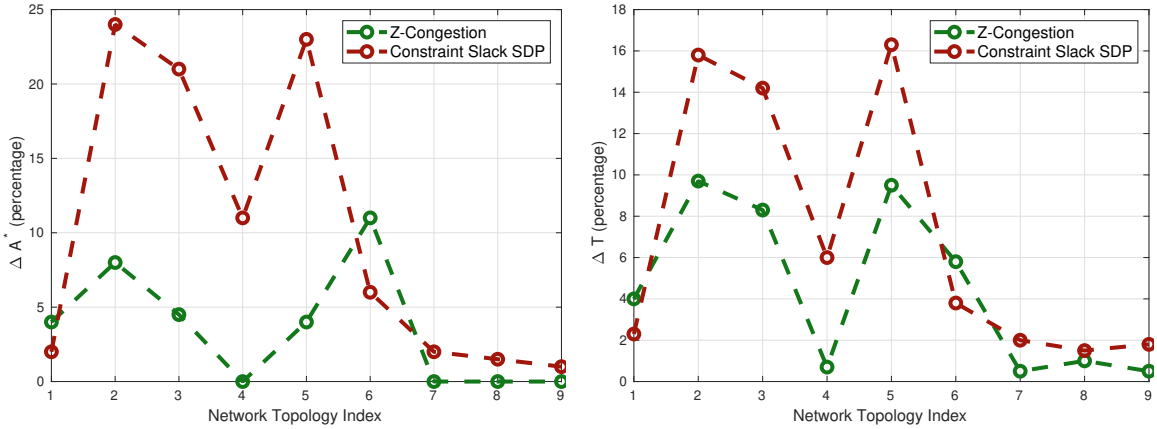
3.6.3 Simulation set 3

The previous simulation results indicate 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 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 different *network topologies* represented by the following (s_1, s_2, s_3) value combinations: $(2, 2, 4)$, $(4, 2, 2)$, $(2, 4, 2)$, $(2, 0, 6)$, $(6, 0, 2)$, $(4, 0, 4)$, $(0, 0, 8)$, $(0, 8, 0)$, and $(8, 0, 0)$. These combinations are indexed from 1 to 9, respectively. Based on these network topologies, the performance results after running Algorithm 7 are given in figures 3.7 and 3.8 for $M = 30$ and 40, respectively.

Instead of plotting Δu, we plotted ΔA^* and ΔT, which are defined similarly to Δu. More specifically, ΔA^* is calculated by comparing the A^* value after running the algorithm with the original A^* value, and ΔT is calculated by first comparing the T_m value after running the algorithm with the original T_m value for each m and taking the worst violation.

As shown in all the figures, Z-Congestion consistently exhibits lower constraint violation than Constraint Slack SDP because Z-congestion achieves a better load-balancing among the ESs. This happens because 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 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; however, this is not the case in general, and is not the case in our simulations.



(a) (b)

Figure 3.8: Performance of Constraint Slack SDP and Z-Congestion, $\epsilon_\tau = 5\%$ and $M = 40$.

3.7 Summary

In this chapter we have considered the problem of DT placement so that application data request timing 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 simulations, it is shown that the Z-Congestion algorithm outperforms the rest in obtaining minimum constraint violation and timing latency.

The DT placement problem can also be approached from another perspective using dynamic network flow models. By viewing the network as a dynamic flow system, we can capture the time-varying nature of data transmission and synchronization processes between PSs and DTs. In the next chapter, we will delve into this alternative viewpoint to solve the optimal DT placement problem, focusing specifically on minimizing the synchronization delay between the PS and the DT. By leveraging dynamic network flow techniques, we aim to develop efficient and scalable solutions that further enhance the performance of DT systems in dynamic network environments.

Algorithm 1 Rounding Algorithm

Input: Fractional solution $\mathbf{X}, \mathbf{Z} \geq \mathbf{0}$ of (SDP-relaxed)

$\mathcal{SM} := \mathcal{M}$
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 algorithm}
  $X_{m,n_1} := X_{m,n_1} + X_{m,n_2}$
  $X_{m,n_2} := 0$
  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**
    $u^*_{m,n} := u_{m,n}$
    if constraint (3.4.19) is violated **then**
      $u_A := \frac{A^* - (T_m + d^{data}_{m,n} + d^{up}_{m,n})}{c_{m,n}}$
      $u_T := \frac{T_m - d^{data}_{m,n}}{c_{m,n}}$
      if $u_T < LHS < u_A$ **then**
        Increase T until $u_T = LHS$
      else if $u_A < LHS < u_T$ **then**
        Increase A^* until $u_A = LHS$
      else if $u_T, u_A < LHS$ **then**
        Increase T and A^* 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 (3.4.23) is violated **then**
      $Z^n_{k,m}, Z^n_{m,k} := X_{m,n}$
    end if
    if constraint (3.4.24) 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

Chapter 4

Digital Twin Placement Using Dynamic Flow Network Evacuation

4.1 Introduction

The placement of DTs on execution servers so that synchronization deadlines are met has been studied in [80]. In this case, the DT synchronization periods are assumed to be provided by the system, i.e., as tolerated latencies in the PS-DT interactions. In this chapter, we study the problem of placing DTs on execution servers so that the *maximum* synchronization periods of a set of DTs is minimized, while (given) DT communication and computation requirements are met. Given that large-scale applications such as smart-cities [30] require the synchronization of many DTs over a common network and computation infrastructure, their execution server placement is an important objective. We define the Minimum Synchronization Period (MSP) problem as that assigning DTs to a set of execution servers so that the maximum synchronization period amongst all PS-DT pairs is minimized.

The main idea in this work is to reduce the MSP problem to that of the well-studied *flow evacuation problems*. An example of flow evacuation is where a site (e.g, a building or geographical area) needs to be evacuated of its residents in the shortest time possible [11]. Treating the synchronization data transmitted from each PS (and processed by the DT) as the objects to be evacuated, the shortest possible synchronization period is the minimum time needed to complete the evacuation for all PS-DT pairs. This correspondence allows us to apply known techniques for solving the MSP problem. We formulate the problem as a multi-commodity flow graph evacuation problem [40] as follows: (i) each PS-DT pair defines a commodity with the PS as the source, (ii) computation delays are modeled as traffic delays on additional special graph edges, (iii) data transmission and computation are modelled as flow on the resulting flow network, and (iv) edge delays depend linearly on the amount of traffic on a given graph edge. The quickest flow is defined as the flow that routes all demands in the shortest time. Since the flow routing solution may result in a fractional execution server assignment for a given commodity (PS-DT pair), we then apply a well-known [25] approximation algorithm to convert the solution to that of unsplittable flows. This ensures that flow for a given commodity uses a single execution server assignment. This is done so that the latency is not significantly increased.

The main contributions of this work are summarized as follows:

- The MSP problem is formulated as a flow problem, modeling the transmission and computation of each PS-DT pair as a commodity on a flow network with linear edge delays. This models the problem as that of a quickest flow evacuation problem, followed by an unsplittable flow rounding procedure that assigns DTs

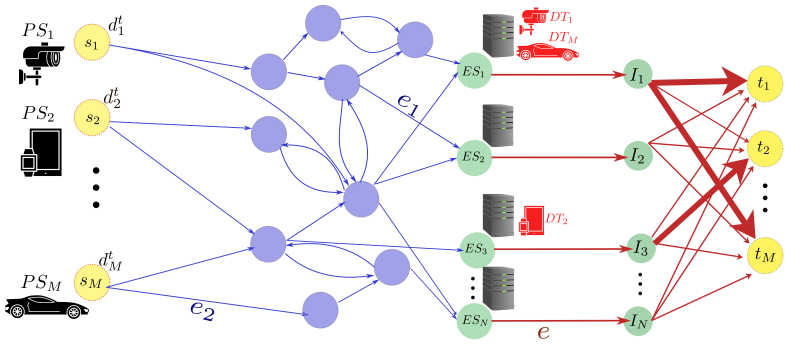


Figure 4.1: System model illustrating the components: PSs (in yellow), network routing nodes (in purple), and ESs along with intermediate nodes (in green). The delay function on edges e, e_1 and e_2 are represented by $\delta_e(f_e)$, $\delta_{e_1}(f_{e_1})$, and $\delta_{e_2}(f_{e_2})$ respectively.

to servers.

- Solutions to the MSP problem are generated both when packet routing follows a single PS-DT path, and when multiple transmission paths are allowed.

- Discrete event simulation results are presented that demonstrate the quality of the MSP solutions produced by our algorithm using the optimal fractional solution of the MSP problem as a lower bound on the optimal integer solution.

The rest of this chapter is organized as follows. The system model is described in detail in Section 4.2. In Section 4.3, we formulate the MSP problem and describe our solution approach. Simulation results are provided in Section 4.4 to illustrate and validate our approach. Finally, we present our conclusions in Section 4.5

4.2 System Model

Figure 4.1 illustrates a networked system in which multiple PSs are connected to a set of ESs through communication nodes. We define $\mathcal{M} = \{1, 2, \ldots, M\}$ as the set of indices for the PSs and $\mathcal{N} = \{1, 2, \ldots, N\}$ as the set of indices for the ESs. Each PS, s_i, is associated with a DT, representing by node t_i (see yellow nodes in Figure 4.1), which must be placed on one of the ESs, ES_j, for $j \in \mathcal{N}$. Each DT is regularly updated with synchronization data by communicating with its associated PS. As PSs transmit synchronization data through the communication links, the data is routed and received by the ESs. After undergoing a series of computations, the processed data becomes accessible to other applications.

Each PS, s_i, in the network has specific demands that must be met to ensure the timely synchronization of DT updates. There are two types of demands, transmission demand (d_i^t) and processing demand (d_i^p), where d_i^t represents the volume of data that must be transmitted from s_i to the corresponding t_i. Moreover, d_i^p specifies the required CPU cycles necessary for processing the data to synchronize the DT at the designated ES, ES_j, where it is placed.

Furthermore, a distance delay L_e is associated with each network link e, accounting for the propagation delay over that link. Each transmission edge e has a transmission rate R_e, determining the time required to transmit data over that link. Each ES ES_j has a CPU frequency F_j, which influences the computational delay for processing the synchronization data.

The main objective of this chapter is to determine the optimal placement of DTs across the available ESs to achieve the minimum overall delay while ensuring that all DTs corresponding to the various PSs are updated. This involves minimizing

both the communication delays (due to data transmission and propagation delay) and computational delays (due to data processing at the ESs) associated with each DT as it periodically refreshes data from its respective PS.

To achieve our objective, we propose a dynamic flow network model of our system to utilize the quickest flow problem for optimizing the placement of DTs. This approach dynamically routes data flow from PSs through network routing nodes to ESs, effectively balancing both communication delays and computational demands. In the following sections, we introduce the dynamic flow network model that serves as the foundation for this optimization problem.

4.3 Problem formulation and solution

We model the transmission network together with the computation that ESs will need to perform for their DTs as a dynamic flow network $G = (V, E)$ with V denoting the set of the nodes and E the set of the edges, where the i-th PS-DT pair corresponds to a commodity i with its own source and sink nodes, and demand d_i^t. Figure 4.1 shows such a network.

The set of nodes V is defined as follows:

- For each PS-DT pair i there is a commodity i, with source and sink nodes s_i and t_i respectively. The demand d_i of commodity i is set to its transmission demand d_i^t, and the i-th PS is placed on node s_i.
- For each ES j there is a pair of nodes ES_j-I_j.
- There is a node for every network routing node.

The set of directed edges E is defined as follows:

- Network links used for data transmission and appear in blue colour in Fig. 4.1: The set of these communication edges is denoted by E_t, and the delay incurred by the transmission of f_e bits on edge $e \in E_t$ is given by $\delta_e(f_e) = f_e/R_e + L_e$.

- Edges connecting each ES_j to its corresponding I_j: The set of these edges is denoted by E_p. Each edge $e = (ES_j, I_j)$ models the computation work done on ES_j for the DTs placed on it. More specifically, if the DT for PS-DT pair (commodity) i is placed on ES_j, the computation delay it incurs on e is $\delta_e^i(f_e^i) = \frac{d_i^p f_e^i}{d_i^t F_j}$, where f_e^i is the fraction of d_i^t arriving at ES_j. Since eventually our solution will collect all d_i^t on the ES j where the i-th DT is placed, in the end $f_e^i \in \{0, d_i^t\}$, and $\delta_e^i(f_e^i)$ will indeed be equal to the computation delay incurred on ES_j due to DT i. The total delay function for edge e is given by $\delta_e(f_e^1, \ldots, f_e^M) = \sum_{i=1}^M \delta_e^i(f_e^i)$.

- Edges (I_j, t_i) for every ES_j and PS-DT pair (commodity) i: There are no delays on these edges.

Note that there are no edge capacities.

We formulate the problem of placing the DTs and routing their transmission demands as a quickest flow problem [13]. As is common in dynamic flow problems, we assume that time is discretized. We denote by $f_e^{i,t}(\theta)$ the flow (of bits) of PS-DT pair (commodity) i transmitted on edge e at time θ. The corresponding delay incurred on the edge is $\delta_e(\sum_{i=1}^M f_e^{i,t}(\theta))$. In addition, it will be useful in the following to also define processing flows $f_e^{i,p}(\theta) = \frac{d_i^p}{d_i^t} f_e^{i,t}(\theta)$. Then the delay on an edge $e = (ES_j, I_j)$ is $\delta_e(f_e^p(\theta)) = f_e^p(\theta)/F_j$, where $f_e^p = \sum_{i=1}^M f_e^{i,p}(\theta)$. Since in what follows we will be dealing only with transmission flows $f_e^{i,t}$, we drop the superscript t.

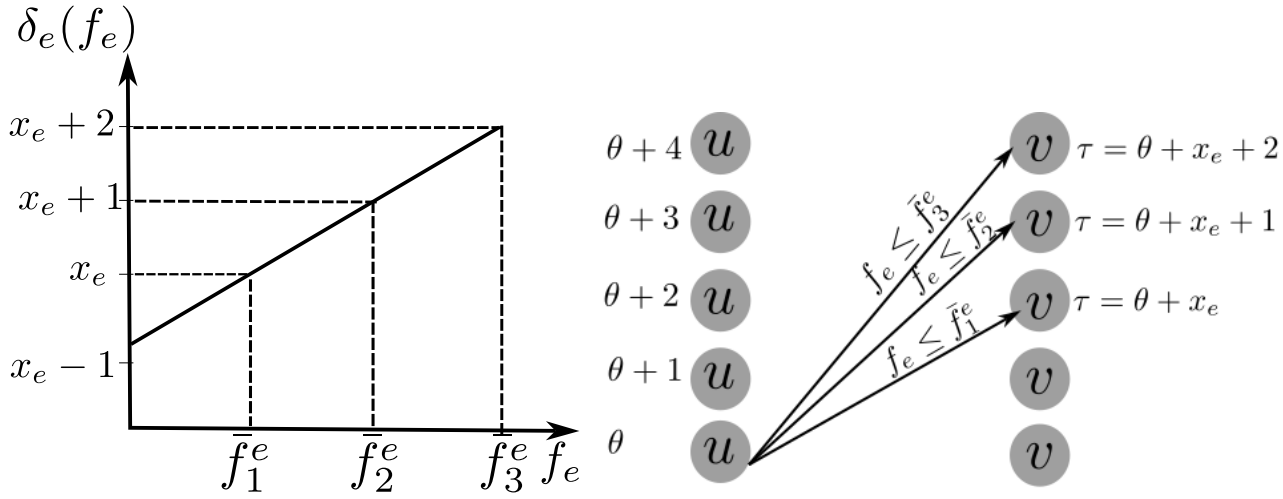


Figure 4.2: The discretized transit times and capacity of time-space links. (a) The transit time function $\delta_e(f_e)$ for link e, where $x_e = 2$. (b) The time-space links corresponding to the link e between nodes u and v.

Our problem departs from the classic quickest flow problem by requiring that a DT is placed on a single ES, i.e., the commodity i flow can enter t_i only through a single edge (I_j, t_i) that forces its DT placement on ES_j. We will consider two versions of the problem: The *single-path* version requires the commodity i demand to be routed to t_i through a single-path from s_i, while the *multi-path* version allows the transmission of demand d_i^t to a single ES through multiple network paths.

4.3.1 Algorithm

The proposed algorithm works in two stages: First, it solves the multi-commodity quickest flow problem on the *time-expanded network* [32] of the "spatial" flow network defined above (cf. Figure 4.1). Next it rounds the solution to get unsplittable flow paths either from each source (single-path version) or just into each sink (multi-path version).

Time-Expanded Network: For a time horizon T, we construct a time-expanded network $G_T = (V_T, E_T)$ from G as follows.

First, if $\delta_e(f_e)$ for an edge e takes integer values $x_e, x_e + 1, x_e + 2, \ldots, x_e + J - 1$ at flow values $\bar{f}_j^e$, $j = 1, \ldots, J$, we define breakpoints $(\bar{f}_j^e, x_e + j - 1)$ on the function graph of δ_e, as shown in Figure 4.2a.

To construct V_T, a copy of the node set V is made for each time $\theta = 1, \ldots, T$, so that for each node v, there are now T nodes $v_\theta \in V_T$. To construct E_T, for each edge $e = (u, v) \in E$, a set of edges $e_{\theta,\tau} = (u_\theta, v_\tau)$ is constructed, joining u_θ to the nodes v_τ at all later times $\tau = \theta + x_e, \theta + x_e + 1, \ldots, T$. The capacity of $e_{\theta,\theta+x_e+j-1}$ for $j = 1, \ldots, T + 1 - (\theta + x_e)$ is set to $\bar{f}_j^e$ (see Figure 4.2b). After constructing G_T, the original dynamic flow problem on G is reduced to a static flow problem on G_T.

LP formulation: Define $f_e^{in}(\theta)$ as the incoming flow for edge $e \in E_T$ at time θ, when $f_e^{in}(\theta)$ lies between two breakpoints $\bar{f}_{\theta,\tau}^e$ and $\bar{f}_{\theta,\tau+1}^e$, it can be written as a convex combination $f_e^{in}(\theta) = \lambda_{\theta,\tau}^e \bar{f}_{\theta,\tau}^e + \lambda_{\theta,\tau+1}^e \bar{f}_{\theta,\tau+1}^e$, where $\lambda_{\theta,\tau}^e + \lambda_{\theta,\tau+1}^e = 1$. Then $\delta_e(f_e^{in}(\theta))$ and $f_e^{in}(\theta)$ can be calculated as follows:

$$\delta_e(f_e^{in}(\theta)) = \sum_{\tau=\theta+x_e}^{T} \lambda_{\theta,\tau}^e \delta_e(\bar{f}_{\theta,\tau}^e) = \sum_{\tau=\theta+x_e}^{T} \lambda_{\theta,\tau}^e (x_e + \tau - \theta), \tag{4.3.1}$$

$$f_e^{in}(\theta) = \sum_{\tau=\theta+x_e}^{T} \lambda_{\theta,\tau}^e \bar{f}_{\theta,\tau}^e, \tag{4.3.2}$$

where (4.3.1) is due to the definition of $\bar{f}_{\theta,\tau}^e$, provided that

$$\sum_{\tau=\theta+x_e}^{T} \lambda_{\theta,\tau}^e = 1, \lambda_{\theta,\tau}^e \geq 0, \quad \tau = \theta + x_e, \ldots, T \tag{4.3.3}$$

and set $\{\lambda^e_{\theta,\tau}\}$ has at most two neighboring non-zero members. Flow $f^{in}_e(\theta)$ is decomposed into flows

$$f^{in}_e(\theta,\tau) = \lambda^e_{\theta,\tau}\bar{f}^e_{\theta,\tau},\ \forall e_{\theta,\tau} \in E_T. \tag{4.3.4}$$

The same approach applies to the outgoing flow $f^{out}_e(\theta)$ of edge e at time θ, to get constraints symmetric to (4.3.3).

The calculation of flow values on G_T is reduced to the calculation of values $\lambda^e_{\theta,\tau} = f^{in}_e(\theta,\tau)/\bar{f}^e_{\theta,\tau}$ by (4.3.4). Edge flows consist of their commodity components, i.e., $f^{in}_e(\theta) = \sum_{i=1}^M f^{in,i}_e(\theta), f^{in}_e(\theta,\tau) = \sum_{i=1}^M f^{in,i}_e(\theta,\tau), f^{out}_e(\theta) = \sum_{i=1}^M f^{out,i}_e(\theta), f^{out}_e(\theta,\tau) = \sum_{i=1}^M f^{out,i}_e(\theta,\tau)$. Recall that we consider only transmitted flow, and processing flow on any edge $e \in E_p$ is $f^{in,p}_e(\theta) = \sum_{i=1}^M \frac{d^p_i}{d^t_i} f^{in,i}_e(\theta),\ f^{out,p}_e(\theta) = \sum_{i=1}^M \frac{d^p_i}{d^t_i} f^{out,i}_e(\theta)$. Given a time horizon T, checking whether all demands can be routed within time T is equivalent to checking the feasibility of the following LP [13]:

$$\sum_{e\in\text{in}(v)} \sum_{\tau=1}^{\theta-x_e} f^{in,i}_e(\tau,\theta) - \sum_{e\in\text{out}(v)} \sum_{\tau=\theta+x_e}^{T} f^{out,i}_e(\theta,\tau) = 0, \tag{MQFP}$$

$$\forall v \in V \setminus \{s_i,t_i\}, \forall i \in \mathcal{M}, \forall \theta \in \{1,\dots,T\} \tag{4.3.5}$$

$$\sum_{e\in\text{in}(t_i)} \sum_{\theta=1}^{T} \sum_{\tau=1}^{\theta-x_e} f^{in,i}_e(\tau,\theta) = d^t_i, \quad \forall i \in \mathcal{M}, \tag{4.3.6}$$

$$\sum_{\tau=\theta+x_e}^{T} \sum_{i\in\mathcal{M}} f^{in,i}_e(\theta,\tau)/\bar{f}^e_{\theta,\tau} \le 1, \forall e \in E_t, \theta \in \{1,\dots,T\} \tag{4.3.7}$$

$$\sum_{\tau=1}^{\theta-x_e} \sum_{i\in\mathcal{M}} f^{out,i}_e(\tau,\theta)/\bar{f}^e_{\tau,\theta} \le 1, \forall e \in E_t, \theta \in \{1,\dots,T\} \tag{4.3.8}$$

$$\sum_{\tau=\theta+x_e}^{T} \sum_{i=1}^{M} \left(\frac{d^p_i}{d^t_i} f^{in,i}_e(\theta,\tau)\right)/\bar{f}^e_{\theta,\tau} \le 1, \forall e \in E_p, \theta \in \{1,\dots,T\} \tag{4.3.9}$$

$$\sum_{\tau=1}^{\theta-x_e} \sum_{i=1}^{M} \left(\frac{d^p_i}{d^t_i} f^{out,i}_e(\tau,\theta)\right)/\bar{f}^e_{\tau,\theta} \le 1, \forall e \in E_p, \theta \in \{1,\dots,T\} \tag{4.3.10}$$

$$f^{in,i}_e, f^{out,i}_e \ge 0, \qquad \forall e \in E, \forall i \in \mathcal{M}. \tag{4.3.11}$$

Here, in(v) and out(v) denote the sets of edges entering and leaving node v, respectively. Constraints (4.3.5)-(4.3.6) guarantee flow conservation and demand satisfaction. Constraints (4.3.7)-(4.3.8) encode constraint (4.3.3) for flow incoming to $e \in E_t$ and the symmetric ones for the outgoing flow, and constraints (4.3.9)-(4.3.10) encode the same constraints for edges $e \in E_p$. Note that equalities have been replaced with inequalities, and there is no encoding of the requirement that only two neighboring λs can be non-zero, since it is well-known [13] that a solution will satisfy the latter requirement and constraints (4.3.7)-(4.3.10) with equality.

Fractional placement and routing: The minimum time horizon T_{opt} for which (MQFP) is feasible can be found by binary search (start with $T = 1$, and double T until feasibility is achieved for some value $T_{max} \leq 2T_{opt}$; then do binary search in $[T_{max}/2, T_{max}]$), after solving $O(\log T_{opt})$ instances of (MQFP). Solving (MQFP) can be done in pseudo-polynomial time $O((|V||E|T_{max})^c)$ for a constant c depending on the LP solver; an approximate solution can be found in polynomial time [40], but we do not apply their techniques here.

After the last step, the minimum time horizon T_{opt}, as well as the scheduling of transmitted flow on the edges of G that satisfies all M demands within time T_{opt} are computed. Unfortunately, this (multi-path) scheduling will have to use, in general, more than one ES node for each DT. Therefore, we need to round the flow entering a terminal t_i on a single edge (I_j, t_i), which also implies that all flow for the i-th PS-DT pair is routed on edge (ES_j, I_j) as well, i.e., ES_j is the placement of DT i. In addition, this requirement for unsplittable flow scheduling extends to the whole flow-path used by commodity i in the single-path version of MSP.

Unsplittable flow rounding: The basis of our flow rounding heuristic is the

single-source unsplittable flow algorithm of [25]. Their algorithm rounds a multi-path single-source flow to single-path flows from the (common) source to each commodity terminal, without increasing the total flow on each edge by more than a factor of 2 of its splittable value. This is done by cancelling flow on cycles formed by two flow paths between a network node and a terminal, by routing the flow on one of the paths onto the other path, until the terminal receives its demand through a single edge. Then the terminal is moved upstream that edge, and the process is repeated until all terminals arrive at the (common) source. This *cycle-cancellation* algorithm cannot be applied directly to the solution of (MQFP), because (i) the multi-path flow is dynamic, and (ii) there are different sources for different PS-DT pairs (commodities). Therefore, the proposed algorithm goes through the following steps:

1. *Time-expanded network consolidation:* We collect all incoming flow of a commodity i on the copies $e_{\theta,\tau} \in E_T$ of edge $e \in E$ for all θ, τ on a single flow value $f_e^i = \sum_{\theta,\tau} f_e^{in,i}(\theta,\tau)$. As a result, the flow timing information is ignored, i.e., we revert to a static flow allocation on the original network G.

2. *Cycle cancellation:* We apply the cycle-cancellation of [25] separately on each commodity either until its demand is routed through a single ES (multi-path version) or, until there is also a single flow path from the source (single-path version).

Transmission and computation scheduling: Once the transmission flow values on G are known, the actual delays (flow incoming and outgoing times for each edge) can be easily computed using the delay functions $\delta_e(f_e)$. The algorithm does not need to do these calculations once it has made all routing decisions, but we perform them for our simulation results in Section 4.4.

Algorithm 4 MSP

Input: $G = (V, E)$ and d_i^t, d_i^p, $\forall i \in \mathcal{M}$
Output: DT placement and routing of d^t on the network
Binary search on T solving (MQFP) on G_T, to determine T_{opt}, fractional placement, and flow routing $f_e^i(\theta),\ \forall i \in \mathcal{M}, e \in E_T, \theta \in \{1, \ldots, T_{opt}\}$
Consolidate flow f_e^i on $G_T = (V_T, E_T)$, to get flow $\hat{f}_e^i,\ \forall i \in \mathcal{M}, e \in E$
for $i \in \mathcal{M}$ **do**
    Cancel cycles [25] to get unsplittable flow $\tilde{f}^i$ from $\hat{f}^i$
end for
for $i \in \mathcal{M}$ **do**
    If $\tilde{f}^i$ uses ES_j, place DT i at ES_j
end for
Output $\tilde{f}^i,\ \forall i \in \mathcal{M}$

The algorithm is summarized in Algorithm 4.

4.4 Simulation Results

In this section, we evaluate the performance of our proposed algorithms through comprehensive computer simulations. We implemented the optimization problem defined in (MQFP) using Python and solved it with the Gurobi optimization toolbox [38].

To ensure statistical robustness, each data point in the presented figures represents the average result of 10 simulation runs on a single network topology. The network topologies are constructed with varying numbers of PS-DT pairs and ESs to assess the scalability and effectiveness of our algorithms under different conditions. Specifically, the number of PSs M ranges from 4 to 10, and the number of ESs N varies between 3 and 6. In addition to the PSs and ESs, the network includes 40 routing nodes that are randomly interconnected to form the communication infrastructure.

To construct the network topology, three layers of nodes in the network are considered: PSs, network routing nodes, and ESs. First, we randomly connect 40 routing

nodes to each other to form the underlying network topology. Next, each ES is randomly connected from 1 to 3 of these routing nodes, ensuring multiple paths exist from PSs to ESs. Each PS is also randomly connected to one of the routing nodes. For generating these random network topologies, we use the NetworkX library in Python [39], which provides tools for creating and manipulating complex networks.

The CPU cycle demand d_i^p for each processing task is randomly selected from a uniform distribution between 10^8 and 10^9 cycles. The transmission data demand d_i^t, is fixed at 25 Mbits for all PS-DT pairs. Each ES j is assumed to have a processing speed F_j of 2.5 GHz [42]. The transmission speed R_e on the edges between PSs and network routing nodes is randomly chosen between 1 Gbps and 5 Gbps [80], reflecting typical wireless access network speeds. For the edges within the core network—including those between network routing nodes and between network routing nodes and ESs—we assume a 5G network backbone supporting bit rates ranging from 5 Gbps to 10 Gbps under stable and uncongested network conditions [6]. Additionally, we account for link processing overhead to better approximate real-world network performance. Propagation delays L_e are calculated based on the physical distances between nodes. For the edges between PSs and network routing nodes, the distances are randomly selected between 1 km and 10 km. Using a propagation speed of 2×10^8 m/s (the approximate speed of light in optical fiber or typical wireless propagation speeds), the resulting propagation delays L_e range from 5 μs to 50 μs.

To validate the effectiveness of our proposed algorithms, we compare the theoretical delays calculated using our delay functions $\delta_e(f_e)$ with the actual delays observed in discrete event simulations. The simulations are conducted using the SimPy library [4], which allows us to model the network components and their interactions

over time, capturing the dynamic behavior of data flows through the network. By simulating the transmission and processing events, we obtain the actual delays experienced by the data packets.

In order to evaluate the routing and placement solutions provided by Algorithm 4, we calculated the actual delays of transmitting and processing the data demands d_i^t for all $i \in \mathcal{M}$ using the discrete event simulation library SimPy [4]. The simulation models the network components and their interactions over time, accurately capturing the dynamic behavior of data flows through the network. By simulating the transmission and processing events, we obtained the actual delays experienced by the data packets. We compared these simulated delays with the delays computed using our delay functions $\delta_e(f_e)$. The results indicate that our calculated delays closely match the actual delays observed in the simulations, affirming that our approximations effectively capture real-world dynamics.

In Figures 4.3 and 4.4, we compare the total delay for transmitting the data d_i^t across three different ESs. The total delay presented is the maximum total delay among all PSs. The figures also show the optimal solution, which is calculated by exhaustively searching over all possible DTs placements on the ESs.

In Figure 4.3, we plot the delay values obtained through discrete event simulation, showing results for both multi-path and single-path transmission schemes. The figure demonstrates that the multi-path scheme consistently outperforms the single-path approach in terms of reducing total delay. The multi-path approach distributes the transmission load across multiple transmission routing nodes, minimizing bottlenecks and improving transmission efficiency.

In Figure 4.4, we compare the delay values as a function of the data demand

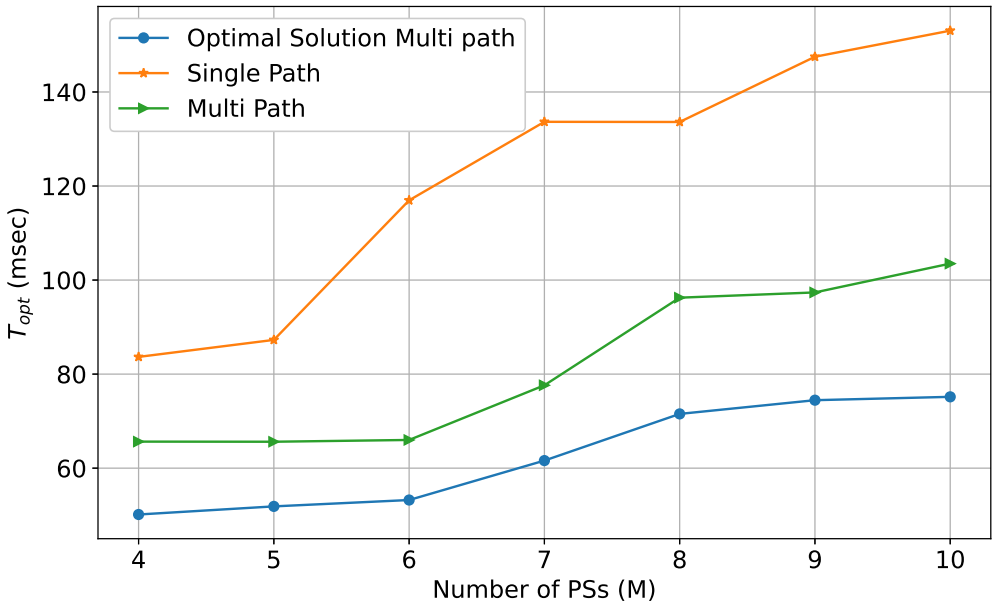


Figure 4.3: Total delay versus number of PSs.

for each PS (in Mbits), that we have 10 PSs in total, using discrete event simulations. Assuming $N = 4$, which is the number of available ESs, this figure shows that the multi-path transmission scheme consistently achieves lower total delays than the single-path approach, particularly as demand increases. However, the multi-path delays remain higher than the optimal solution.

Table 4.1 compares the total delays (in milliseconds) for different network topologies characterized by the number of ESs in each of the three groups with low (G_1), which operate, moderate (G_2), and high (G_3) processing rates. ESs in G_1 operate at 1 GHz, G_2 at 2.5 GHz, and G_3 at 5 GHz. We compare the performance of Algorithm 4 and its single-path version against the optimal delay across various configurations. The network topologies considered are represented by the combinations of (G_1, G_2, G_3) as follows: $(2, 2, 4)$, $(0, 4, 0)$, $(4, 0, 0)$, $(2, 0, 2)$, $(2, 2, 0)$, and $(2, 1, 1)$.

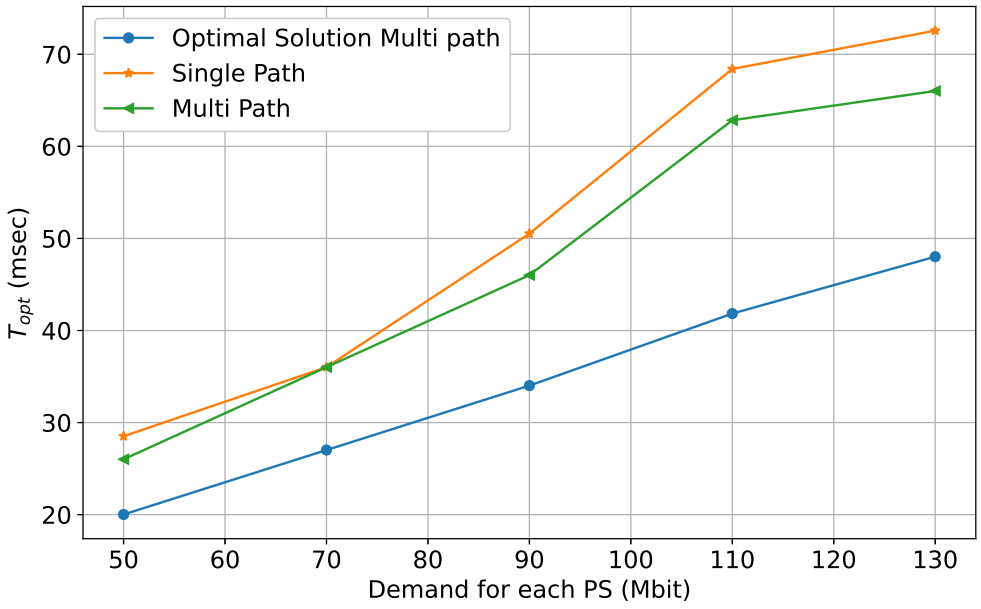


Figure 4.4: Total delay versus demand of each PS (d_i^t).

The results indicate that Algorithm 4 consistently achieves lower total delays compared to its single-path version and approaches the optimal delay, demonstrating its effectiveness in handling heterogeneous processing rates among ESs.

Table 4.1: Comparison of total delays (in milliseconds) for different network topologies

Network Topology (G_1, G_2, G_3)	**Multi-path Algorithm**	**Single-Path Algorithm**	**Optimal Solution**
(0, 4, 0)	29.97	40.69	20.68
(4, 0, 0)	120.40	120.49	88.35
(2, 0, 2)	40.54	48.14	33.08
(2, 1,1)	31.43	33.17	24.12
(1, 2, 1)	23.74	40.69	10.08
(2, 1, 1)	32.56	45.66	22.85

Overall, the simulation results demonstrate the robustness and effectiveness of our proposed algorithms in minimizing synchronization periods for DTs in a networked

environment. The close agreement between the theoretical and simulated delays validates our modeling approach and confirms the practical applicability of our solutions.

4.5 Summary

In this chapter, we address the Minimum Synchronization Period (MSP) problem, which focuses on optimally placing digital twins (DTs) on execution servers (ESs) and routing synchronization data from physical systems (PSs) to DTs to minimize the maximum synchronization period among all PS-DT pairs. Mapping the MSP problem to flow evacuation problems, we formulated it as a multi-commodity quickest flow problem on a flow network with linear edge delays. This approach allowed us to leverage existing techniques to determine the quickest way to route all synchronization data while meeting communication and computation constraints. To ensure that each DT is placed on a single ES, we applied an unsplittable flow rounding procedure. This method effectively reroutes the data flows without significantly increasing the maximum synchronization period, thus providing a practical solution for DT placement and data routing. Algorithms were developed for both single-path and multi-path data transmission scenarios, offering flexibility depending on network capabilities and requirements. Through discrete event simulation, we demonstrated the effectiveness of our proposed algorithms. The simulation results confirmed that our approach produces high-quality MSP solutions, closely approximating the optimal synchronization periods.

Chapter 5

Age of Information in Vehicular Digital Twin Migration

5.1 Introduction

Operating a DT typically involves real-time synchronization between it and the PS. This involves periodic communication between the two, so that any changes in the state of the PS are known by its DT. Using this information, the DT may provide *features* to external applications that reflect the operation of the PS [78]. Examples of feature information might include whether an electric circuit is working normally, or whether a mechanical subsystem requires maintenance, etc. Both the data synchronization and data processing at the DT introduce latency, and as a result, the available feature information at the DT has an associated AoI. This is defined as the amount of time elapsed since the state data used to obtain the feature was generated at the PS. In order to keep tight synchronization between the DT and its PS, a DT is usually located close to the PS to reduce the data transmission delays [78].

In vehicular networks, DTs can be created for individual vehicles [99], human drivers [41], or an entire connected vehicle system [83]. These DTs can be leveraged to make driving decisions in autonomous driving, improve driving performance of human drivers, or support applications for vehicular users. The AoI of these DTs can be adversely affected by the high mobility of the vehicles. When a vehicle crosses a cellular boundary for example, the synchronization latency between the vehicle and its DT can increase significantly [17]. In this case it may be desirable to move the location of the DT to a more appropriate server so that the AoI at the DT is improved. This is referred to as *DT migration*.

Determining the time to initiate DT migration can be challenging. In addition to security issues [100, 98, 56], the AoI at the DT during and after migration should be taken into consideration when deciding the best time to initiate the migration. Migrating a DT may require a considerable amount of time, since it involves moving not only the computation models used to process PS inputs but also the historical data needed to extract feature information. To our knowledge, no prior work has been done to study the effect of DT migration on AoI at the DT.

In this chapter, we study the problem of migrating the DT of a mobile PS from its current server to a server with faster data updates, so that the expected AoI of the data provided by the DT after a random request by an application is minimized. The problem is captured by the following typical scenario: A mobile (e.g., vehicular) PS follows a trajectory that crosses from its current cellular domain to a different one. As a result, the communication between PS and its DT installed at a server in the current domain will become significantly slower once the PS crosses to a different domain. We study the migration of the DT to a new server in anticipation of this

cross-over, so that the 'freshness' of the DT data as measured by their time-averaged AoI is optimized, and the problem is reduced to the determination of the optimal initiation time for migrating the DT based on known statistical information of the PS movement.

The problem is formulated as a Markov decision process and we propose an online algorithm to find the optimal migration initiation time using optimal stopping theory and dynamic programming. Several other algorithms are also proposed and their performance is compared with the online optimal algorithm as well an offline algorithm that gives a lower bound on the average AoI. The main contributions of this chapter are summarized as follows.

1. Vehicular DT migration costs are defined based on the average AoI that is sampled at the digital twin. This definition incorporates the PS-DT synchronization and the latencies associated with the migration process when the PS crosses cellular boundaries.

2. We formulate the problem of determining the optimal migration initiation time as an optimal stopping problem. This allows us to account for available information at the beginning of the vehicle's journey and to minimize the cost of migration within a stochastic vehicular traffic environment.

3. We present an online optimal migration initiation decision algorithm that meets migration time deadline constraints but also achieves the minimum cost of migration.

4. A best-in-expectation algorithm is introduced that offers a sub-optimal yet

faster solution. This algorithm is beneficial in scenarios where the computational complexity of dynamic programming (DP) is too high. The algorithms are also compared with two heuristics that do immediate migration and migration at handoff.

5. We formulate an offline computation for Age of Information that provides a theoretical lower bound on AoI performance. This is used for comparisons with the other algorithms.

The rest of this chapter is organized as follows. First, we review some related work in Section 5.2. Then in Section 5.3, we describe the system model in detail, followed by a thorough description of PS-DT synchronization and an analysis of the handoff and migration decision process. This concludes with a formulation of the optimal stopping problem which leads to the optimal migration time criteria in Section 5.4. Following this, we employ dynamic programming to propose an online optimal migration algorithm in Section 5.5. In Section 5.6, performance results are presented that compare the online optimal algorithm with various other computation migration algorithms. Finally, we present our conclusions in Section 5.7.

5.2 Related Work

Existing networking-related work on DTs mainly falls into two categories. The first focuses on how to leverage DTs of network entities for managing and optimizing network operations and services, e.g., [21, 93, 31, 97], and the second deals with issues for running DTs with desired quality under network resource constraints, e.g., [80, 55, 14, 15]. Among this work, the AoI is an important performance metric, including

the AoI of the feature information kept at a DT. Maintaining low AoI at the DT is beneficial to the applications, but this comes at the cost of consuming more network resources. In [67], the tradeoff between the time-average AoI target and the energy consumption for maintaining the DT is studied through device scheduling and channel allocation. A network may have to support a large number of DTs. The work in [80] and [55] study the DT placement in network servers, taking into consideration the PS-DT and DT-application latency. Random network conditions result in uncertainty in the PS-DT update completion time, which affects the AoI update time at the DT. References [67] and [49] consider the question of whether the DT should respond to a request with the current information or delay until the next PS-DT update completion.

In vehicular networks, DTs have been created for enhancing network performance [96, 97], improving driving performance [83, 31], and creating immersive applications for vehicular users [99]. The high mobility of vehicles causes difficulties in managing network resources and optimizing network operations. DTs are created for vehicles in [96] for a software-defined vehicular network. These DTs enable the network to generate adaptive routing policies in real-time when there are dynamic topology changes. A DT network is maintained in [97] for the network entities in a vehicular edge computing network. By predicting and analyzing network conditions, the DT network helps determine efficient offloading strategies and ensures optimal resource allocation. A DT framework is developed in [83] where the DT can aggregate and process sensed data from the vehicular network entities, including vehicles, drivers, passengers, the road, etc., and the results are used to assist drivers to make

more intelligent driving decisions. In addition, DTs provide a safe virtual environment for risk assessment and enable in-depth analysis through AI integration, which is particularly beneficial. In [31], the virtualization and offline learning capabilities of the DTs is leveraged to achieve safe and efficient lane-changing in connected and automated vehicles. The concept of "vehicular metaverse", or "vetaverse", is introduced in [99], which is an immersive environment that provides diverse and personalized in-vehicle services for vetaverse users. Such services take advantage of vehicular DTs, pedestrians, roads, and traffic signals that allow real-time visualization and observation of the network conditions.

The high mobility of vehicles makes it desirable to migrate their DTs in order to reduce the PS-DT communication delay. Different vehicular DT migration strategies are described in [100] for an autonomous driving scenario, where a data center is responsible for generating and managing the DT of a vehicle, and a copy of the DT model is kept at the server close to the vehicle to maintain low PS-DT communication latency. As a vehicle moves, the migration of its DT can be initiated by the data center, the current edge server, or the vehicle itself. DT migration is also considered in [55] with an objective to optimize a utility function that takes into consideration the data transmission overhead, communication latency, the service quality and experience for users. Transfer learning is used to cope with dynamic network conditions when solving the optimum DT migration problem. In [94], the DT migration problem is addressed in mobile edge computing networks with the aim of minimizing accumulative service costs, including DT update costs, consumer service costs, and migration costs between cloudlets. Similarly, [20] explores a learning-powered social-aware orchestration for the mobility of Social IoT devices and their corresponding Social DTs

. By employing deep learning algorithms for mobility prediction, the study aims to make migration decisions to minimize PS-DT communication latencies.

5.3 System Model

A system is considered where vehicles move along a single-lane highway as shown in Figure 5.1. The vehicles communicate with cellular base stations (BSs) that are divided into geographical domains, each of which has one or more BSs. Within each domain, there is an edge server (ES) connected to the BSs through high speed communication links. We use ES_i to denote the ES located in domain D_i. The DT for each vehicle is hosted at an ES, referred to as the hosting ES for its DT. We consider a typical vehicle, referred to as a PS below, and study the synchronization updates at its DT, the procedure to migrate the DT from one ES to another, and the AoI changes at the DT. Figure 5.1 shows a sample of a vehicle that is at location x_s in D_i at time t_s, then crosses a cellular domain boundary at location $x_{b_{i+1}}$ at time $t_{b_{i+1}}$. Further details of the migration process are discussed below. The system time is discretized into equal duration time slots that are sufficiently short so that the changes to PS mobility information within one slot can be neglected.

5.3.1 PS-DT Synchronization and Age of Information

In order for the DT to act on behalf of the PS, the PS periodically sends an update of its current state to the DT for processing. This is referred to as *PS-DT synchronization*. An example of synchronization activity is shown in Figure 5.2 for the PS and the DT located at ES_i. The two time lines at the top of the figure show four

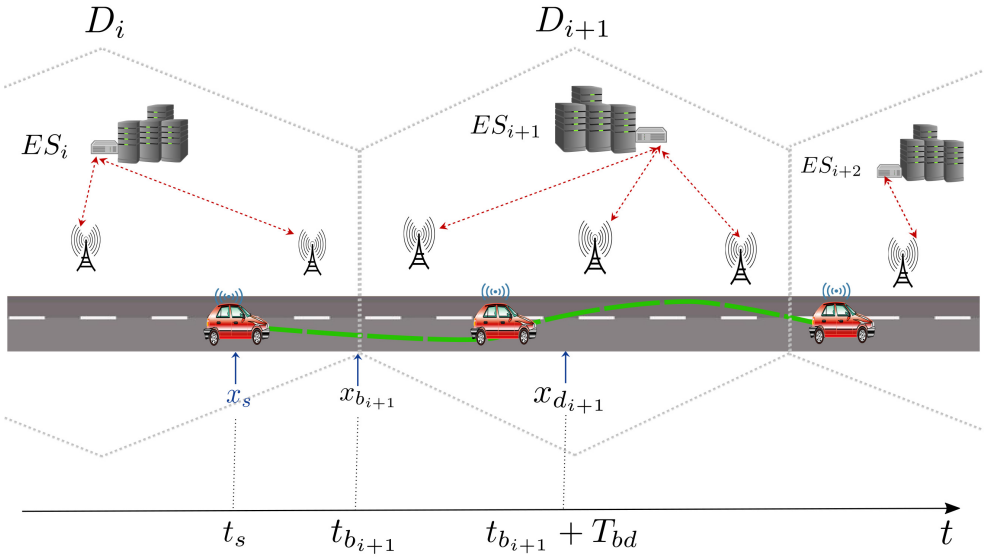


Figure 5.1: Vehicle trajectory crossing a cellular domain boundary $x_{b_{i+1}}$ at time $t_{b_{i+1}}$

periods of PS-DT updates. Each update propagates from the PS to the ES followed by processing at the ES to incorporate the new update. Each update involves the data transmission delay from the PS to the ES, defined as T_{r_i}, and the data processing delay needed at the ES to incorporate the new information into the DT, defined as T_{c_i}. P is defined to be the period of the updates as shown in the figure.

The DT will continuously provide PS features that are available to external applications. The age of information at time t is defined as AoI(t), which is the time that has elapsed since the PS-DT update was initiated that was used to generate the current DT output. An example of the evolution of AoI(t) is shown in Figure 5.2. At time t_0, for example, the DT has just finished processing the first PS-DT synchronization update and AoI(t_0) $= T_{r_i} + T_{c_i}$, i.e., the elapsed time since the update was generated. AoI(t) then increases linearly (at 1 sec/sec) until the next update

has been processed when the AoI drops from $T_{r_i} + T_{c_i} + P$ to $T_{r_i} + T_{c_i}$. Note that we assume that transmit power control is used on the cellular links so that T_{r_i} is a fixed (and known) value. Similarly, we consider that a dedicated portion of the ES execution capacity is made available for the DT so that T_{c_i} is also known and fixed. The methodology introduced in this chapter can be applied to more complex cases, e.g., if one conditions on the parameter values used, one can compute the average AoI performance for random cases by unconditioning each case using the probability of their occurrence. Furthermore, we assume that P is large enough, i.e., $T_{r_i} + T_{c_i} \leq P$, so that there is no queueing delay when a PS and its DT are in the same domain [80].

When the PS is in domain D_i and the DT is located in a different domain, e.g., ES_j for $i \neq j$, the PS update will be forwarded through the cellular backhaul network to the DT in domain D_j. This typically involves much longer transmission delays compared to when the PS and DT are in the same domain. Denote the data transmission delay for a PS in domain i to a DT in domain j to be $T_{r_{i,j}}$, which is assumed to be fixed and known.

5.3.2 DT Migration

In this section we consider the case where a vehicle's cellular domain handoff leads to the need to change the DT location, i.e., *DT migration*. Define t_s as the start time and the location of the PS at $t = t_s$ is x_s as shown in Figure 5.1. Let $x_{b_{i+1}}$ be the boundary between domains D_i and D_{i+1}, and the time that the PS arrives at $x_{b_{i+1}}$ and hands off to D_{i+1} is $\mathbf{t}_{b_{i+1}}$. As the PS moves, the migration of its DT may happen before or after the PS passes the cellular domain boundary. Note that in some cases, it may be better to keep the DT in the previous domain after the PS hands off to a

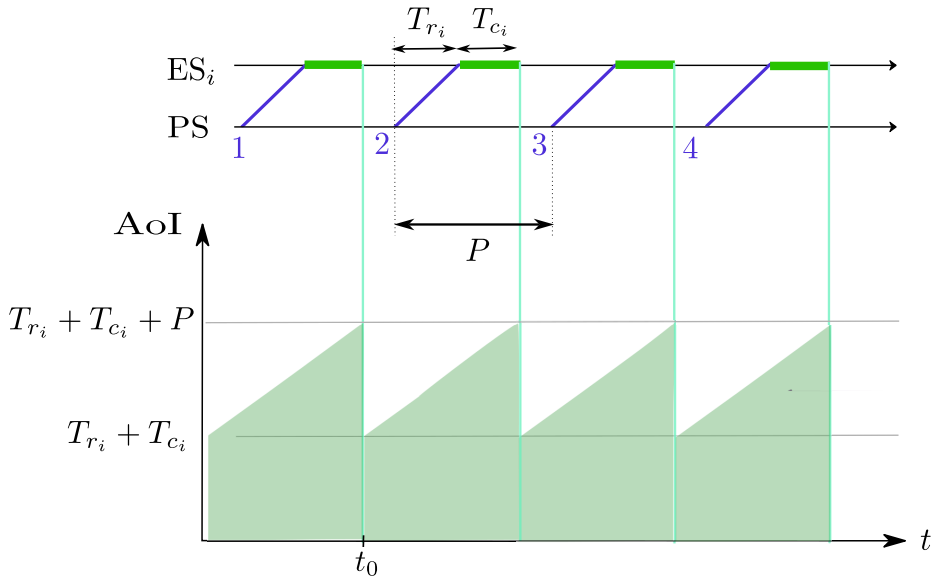


Figure 5.2: AoI of a DT at server ES_i during four synchronization update periods. The upper two timelines show each PS transmission (in blue) followed by the DT processing update (in green).

new domain. This might happen, for example, if the processing time at the ES in the new domain is very long. We assume that on the PS route inside domain D_{i+1}, there is a location $x_{d_{i+1}}$ (shown in Figure 5.1) beyond which migrating the DT from D_i to D_{i+1} is not considered, since its position from the boundary is enough to become irrelevant. We define the time that the PS reaches location $x_{d_{i+1}}$ as $\mathbf{t}_{d_{i+1}}$ and further define $\mathcal{T}_{bd_{i+1}} = \mathbf{t}_{d_{i+1}} - \mathbf{t}_{b_{i+1}}$. Due to the statistical nature of the PS motion, both $\mathbf{t}_{b_{i+1}}$ and $\mathbf{t}_{d_{i+1}}$ are random, so is $\mathcal{T}_{bd_{i+1}}$. In the remaining part of the chapter, we consider a typical DT migration problem when the PS moves from domains D_i and D_{i+1}. We drop the subscript "$i+1$" from $\mathbf{t}_{b_{i+1}}$, $\mathbf{t}_{d_{i+1}}$, and $\mathcal{T}_{bd_{i+1}}$ when there is no confusion, and use t_b, t_d, and T_{bd}, respectively, to denote the values that random variables $\mathbf{t}_b$, $\mathbf{t}_d$, and $\mathcal{T}_{bd}$ take.

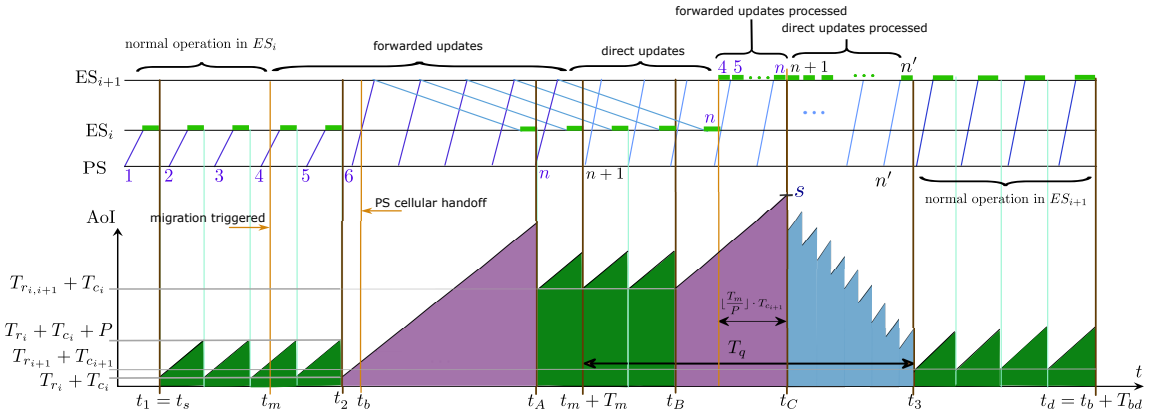


Figure 5.3: AoI when $t_m \leq t_b \leq t_m + T_m$

In this chapter we present a methodology for determining the optimal online DT migration initiation time so that the age of information seen by random arrivals to the DT is minimized. This is a difficult problem and we consider the case where the DT computation and communication times within a given cellular domain are given and constant. In the following, we use the scenario shown in Figure 5.3 to illustrate the timing of different events that may occur as a result of the DT migration process. As before, the timelines at the top show PS-DT synchronization update activity for a PS that is initially updating its DT located in ES_i and initiates a DT migration that moves the DT to ES_{i+1}. We denote by t_m the starting time of the migration, which can be seen to occur during the 4th update interval on the PS timeline in Figure 5.3. Prior to that, the PS is engaged in normal operation in ES_i and 3 update periods (1, 2 and 3) are shown in the figure. The AoI during that time is given by and evolves as discussed in Figure 5.2.

The migration process is summarized as follows. Once the migration has been initiated, ES_i then starts the transfer of the DT to ES_{i+1}. During this time, the current DT (at ES_i) continues to function as the DT and processes updates from the

PS and requests from external applications. These updates are also enqueued and forwarded to the new DT (at ES_{i+1}) where they are eventually applied as updates at the new DT. Once this is finished, the new DT (at ES_{i+1}) becomes active and it processes all DT updates that have occurred in the meantime. To properly assess the effects of migration on AoI, we will describe in detail, all of the interactions that occur as the DT is moved while supporting ongoing PS-DT updates and external application interactions. This is done so that we can properly model and minimize the negative effects of migration on AoI with negligible downtime. The migration can be implemented via standard mechanisms, such as virtual machine or any other running process migration.

To migrate the DT from ES_i to ES_{i+1} and enable ES_{i+1} to process the PS data, ES_i should first transfer the computation models used for processing the PS data, along with any historical data needed to extract feature information, to ES_{i+1}. We define T_m as the time needed to complete this data transfer. This includes the transferring of the operating DT to ES_{i+1} including the forwarding of all the PS-DT updates as discussed above. The example in Figure 5.3 shows the case where $t_m \leq t_b \leq t_m + T_m$, i.e., the migration is triggered (at t_m) *before* the cellular domain handoff (at t_b), which occurs *before* the T_m duration expires. The duration of T_m spans from the 4th update interval and expires during the nth update interval. Other scenarios will be analyzed in Section 5.4. The migration process is discussed in more detail below for the different time ranges marked in the figure.

- $t \in (t_1, t_2)$: Both the PS and the DT (at ES_i) are in domain D_i. The AoI increases (at 1 sec/sec) to the maximum value $T_{r_i} + T_{c_i} + P$ and immediately drops to the minimum value $T_{r_i} + T_{c_i}$ when the DT finishes processing an update.

The AoI then repeats the same pattern as was described in Figure 5.2. At t_m the DT migration is initiated and ES_i starts the transfer of the DT to ES_{i+1}. The DT continues to process PS updates, which are also forwarded to the new DT (at ES_{i+1}) for processing. These are referred to as "forwarded updates" in the figure. This permits the DT (at ES_i) to continue to update the AoI during the migration process.

- $t \in (t_2, t_A)$: The PS crosses the cellular domain boundary at $t = t_b$. In the figure, t_b is shown slightly larger than t_2, and the DT still resides in ES_i. A cellular handoff has just occurred and the AoI increases from the minimum value $T_{r_i} + T_{c_i}$ to the maximum value $T_{r_i} + T_{c_i} + T_{r_{i,i+1}} + T_{c_i}$, and then immediately drops to the new minimum value $T_{r_{i,i+1}} + T_{c_i}$ at $t = t_A$. This drop occurs when the DT at ES_i finishes the first update after the PS enters domain D_{i+1}. Note that during this period, the updates originating at the PS (now in D_{i+1}) are routed back from D_{i+1} to the DT that is in D_i (at ES_i). For simplicity, these updates are shown in Figure 5.3 as intersecting the ES_{i+1} timeline but they are routed by the cellular backhaul network. Note that we assume that the handoff delay has a negligible effect on the AoI.

- $t \in (t_A, t_B)$: The PS is in domain D_{i+1} and the DT is in ES_i. The AoI increases from the minimum value $T_{r_{i,i+1}} + T_{c_i}$ to the maximum value $T_{r_{i,i+1}} + T_{c_i} + P$ and then immediately drops to the minimum value $T_{r_{i,i+1}} + T_{c_i}$ when the DT finishes processing an update. The AoI then repeats the same pattern.

- $t \in (t_B, t_B + P)$: The PS is in domain D_{i+1} and the DT is in ES_i. The DT processes the last update that was generated during the T_m period. The AoI

increases from $T_{r_{i,i+1}} + T_{c_i}$ to $T_{r_{i,i+1}} + T_{c_i} + P$.

- $t \in (t_B + P, t_C)$: The PS is in domain D_{i+1} and the active DT is still in ES_i. The DT (in ES_{i+1}) reprocesses the updates generated during T_m. Since the DT in ES_i is still active, the AoI continues to increase from $t = t_B + P$ throughout this interval. The number of the reprocessed updates is equal to $\lfloor \frac{T_m}{P} \rfloor$, therefore, the reprocessing lasts for $\lfloor \frac{T_m}{P} \rfloor \cdot T_{c_{i+1}}$. At $t = t_C$, the AoI reaches a value of $s = T_{r_{i,i+1}} + T_{c_i} + P + \lfloor \frac{T_m}{P} \rfloor \cdot T_{c_{i+1}}$.

- $t \in (t_C, t_3)$: At time t_C the new DT at ES_{i+1} now becomes the active DT and the original DT in ES_i is terminated. It now processes the queued updates (referred to as "direct updates" in the figure) that have arrived since $t = t_m + T_m$. During this period, the AoI drops by P after each $T_{c_{i+1}}$ interval until $t = t_3$ when there are no remaining updates at ES_{i+1}. The interval shown as T_q in Figure 5.3 gives the time over which the forwarded and direct updates have been enqueued for processing by the new DT.

- $t \in (t_3, t_b + T_{bd})$: The DT at ES_{i+1} is now operating normally in D_{i+1}. As in Figure 5.2, the AoI increases to the maximum value of $T_{r_{i+1}} + T_{c_{i+1}} + P$ and immediately drops to $T_{r_{i+1}} + T_{c_{i+1}}$ when the DT finishes processing a new update.

We define the migration cost $Cost(t_s, t_m, t_b, T_{bd})$ as the average AoI over the period from time t_s to $t_b + T_{bd}$, i.e.,

$$Cost(t_s, t_m, t_b, T_{bd}) = \frac{\int_{t_s}^{t_b + T_{bd}} AoI(t) dt}{t_b + T_{bd} - t_s}. \tag{5.3.1}$$

This is also the expected AoI seen at the DT by application requests that arrive uniformly over the same time period. In (5.3.1), the integral on the right-hand side is the area of the AoI curve over the considered time interval as shown by the shaded areas in Figure 5.3.

5.4 Problem Formulation

In this section, a formulation of the DT migration problem is presented. Recall that we seek to decide on-the-fly the best time for migrating the DT of a moving PS from the server servicing a domain D_i to a server servicing domain D_{i+1} (or decide that no migration is profitable), so that the average AoI of the DT data is minimized. In what follows, t_m is the decision variable of the problem, i.e., the migration initiation time (where $t_m = \infty$ denotes a decision of no migration).

5.4.1 The offline problem

As given in (5.3.1), the migration cost $Cost(t_s, t_m, t_b, T_{bd})$ that we try to minimize by deciding the DT migration time t_m, is the average AoI over the period from time t_s to $t_d (= t_b + T_{bd})$. Recall (Section 5.3) that the expected AoI of the DT data that an application experiences when its request arrives uniformly at random during the period $[t_s, t_d]$ is equal to the area under the curve of AoI as a function of time, divided by $t_d - t_s$. In what follows, we first compute the integral in the numerator on the right-hand side of (5.3.1) by splitting the AoI function into different sections and computing the area under the AoI function (referred to as AoI areas below) for each section. The following functions are defined for calculating the AoI areas for different

time intervals, where we define

$$\underline{t} = \lfloor t/P \rfloor \tag{5.4.1}$$

as the number of updates generated in an interval of length t.

- $C_1(t_\alpha, t_\beta, T_r, T_c)$: It calculates the area under the AoI curve for a time interval $[t_\alpha, t_\beta]$, during which both the data transmission time (T_r) and the computation time (T_c) remain unchanged. We have

$$C_1(t_\alpha, t_\beta, T_r, T_c) = P(\underline{t_\alpha} - \underline{t_\beta})(\frac{P}{2} + T_r + T_c). \tag{5.4.2}$$

  For example, in Figure 5.3, $C_1(t_1, t_2, T_{r_i}, T_{c_i})$ can be used to find the area under the AoI curve when both the PS is in D_i and the DT is in ES_i, in which case $T_r = T_{r_i}$ and $T_c = T_{c_i}$.

- $C_2(h_1, h_2, w)$: It is the area of one single trapezoid, whose height is the time interval of length w, during which the AoI keeps increasing, and h_1 and h_2 are the AoI values at the beginning and end of the time interval. We have

$$C_2(h_1, h_2, w) = \frac{w}{2}\left(h_1 + h_2\right). \tag{5.4.3}$$

  In Figure 5.3, the areas of the AoI curve in the intervals $[t_2, t_A]$ and $[t_B + P, t_C]$ can be found using this function.

- $C_3(T_q, s)$: It computes the area for the time interval during which the server is dequeueing of queued requests after the DT has migrated. This time interval includes the server processing time (i.e., $T_{c_{i+1}}$) for the all backlogged updates

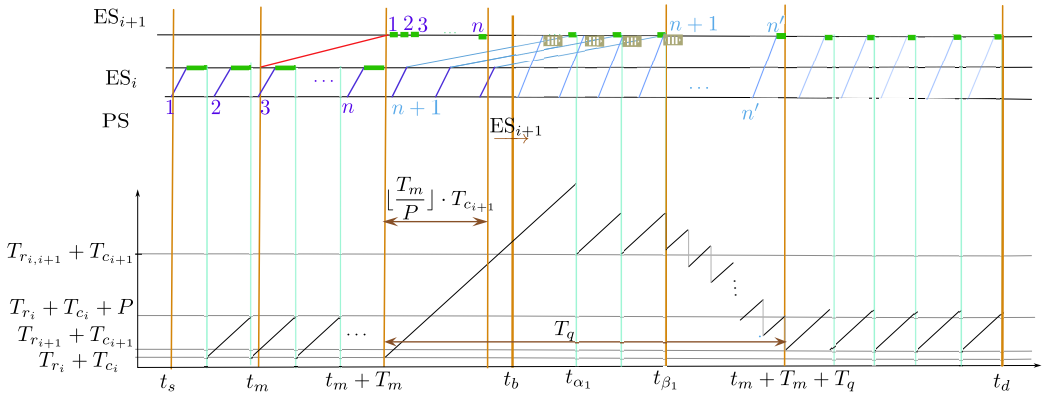
received during T_q. Let s be the AoI at the start of the time interval. The AoI area in the interval is calculated as

$$C_3(T_q, s) = \sum_{n=1}^{T_q+1} \frac{T_{c_{i+1}}}{2}(2s - 2nP + (2n-1)T_{c_{i+1}}). \tag{5.4.4}$$

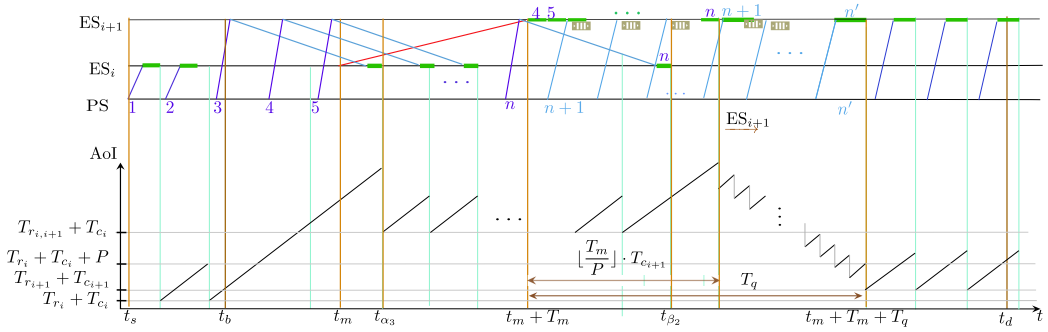
where n represents the index of each queued request. In Figure 5.3, the dequeueing interval of ES_{i+1} is $[t_C, t_3]$.

We make some assumptions that will simplify this computation without sacrificing too much accuracy. We assume that breaking points happen on multiples of P. Also, we ignore updates arriving to the DT during the dequeueing time $[t_C, t_3]$.

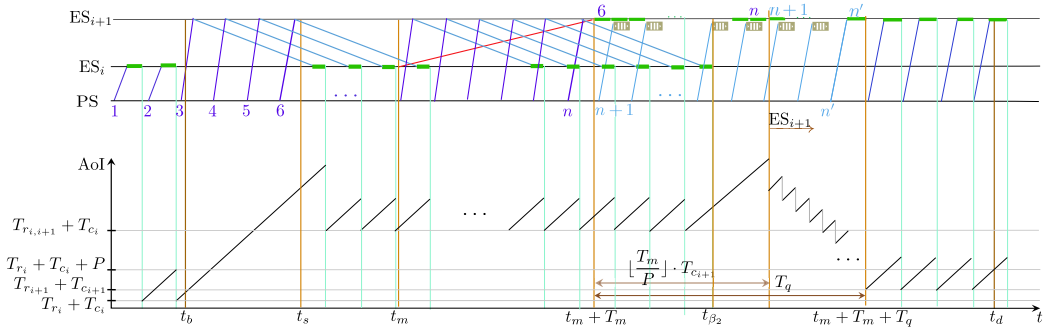
The total AoI area for the interval $t \in [t_s, t_d]$ is calculated based on different cases as shown in Fig. 5.3 and Fig. 5.4.



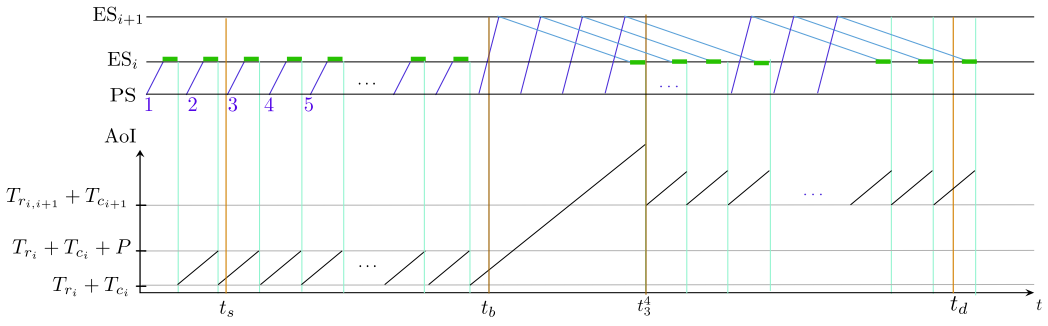
(a) AoI when $t_s \leq t_m + T_m \leq t_b$



(b) AoI when $t_b \leq t_m \leq t_d$:



(c) AoI when $t_b \leq t_s$ and $t_m \leq t_d$



(d) AoI when there is no migration ($t_m = \infty$)

Figure 5.4: Different migration scenarios

Case $t_s \leq t_m + T_m \leq t_b$ as shown in Figure 5.4a. In this case

$$T_q = T_{r_{i,i+1}} + T_{c_{i+1}}. \tag{5.4.5}$$

The AoI area is calculated as

$$\begin{aligned} Area_1(t_s, t_m, t_b, T_{bd}) = \\ & C_1(t_s, t_m + T_m, T_{r_i}, T_{c_i}) + +C_2(h_{11}, h_{12}, w_{11}) + \\ + \quad & C_1(t_{\alpha_1}, t_{\beta_1}, T_{r_{i,i+1}}, T_{c_{i+1}}) + +C_3(T_q, T_{r_{i,i+1}} + T_{c_{i+1}} + P) + \\ + \quad & C_1(t_m + T_m + T_q, t_d, T_{r_{i+1}}, T_{c_{i+1}}) \end{aligned} \tag{5.4.6}$$

with

$$\begin{aligned} h_{11} &= T_{r_i} + T_{c_i}, \\ h_{12} &= T_{r_i} + T_{c_i} + T_{r_{i,i+1}} + T_{c_{i+1}}, \\ w_{11} &= T_{r_{i,i+1}} + T_{c_{i+1}}, \\ t_{\alpha_1} &= \underline{(t_m + T_m)}P + T_{r_{i,i+1}} + T_{c_{i+1}}, \\ t_{\beta_1} &= \underline{t_b}P + T_{r_{i,i+1}} + T_{c_{i+1}}, \end{aligned}$$

and

- $C_1(t_s, t_m + T_m, T_{r_i}, T_{c_i})$ is the AoI area for $t \in [t_s, t_m + T_m]$ during which data transmission is in D_i and data processing is at ES_i;
- $C_2(h_{11}, h_{21}, w_{11})$ is the AoI area for $t \in [t_m + T_m, t_{\alpha_1}]$ during which the AoI monotonically increases due to the inter-domain transmission delay, h_{11} and

h_{12} are the AoI at $t = t_m + T_m$ and $t = t_{\alpha_1}]$, respectively, and w_{11} is the length of the time interval $[t_m + T_m, t_{\alpha_1}]$;

- $C_1(t_{\alpha_1}, t_{\beta_1}, T_{r_{i,i+1}}, T_{c_{i+1}})$ is the AoI area for $t \in [t_{\alpha_1}, t_{\beta_1}]$ during which data transmissions cross the two domains and data processing is performed at ES_{i+1},
- $C_3(T_q, T_{r_{i,i+1}} + T_{c_{i+1}} + P)$ is the AoI area for $t \in [t_{\beta_1}, t_m + T_m + T_q]$ when ES_{i+1} is dequeueing the backlogged updates after DT migration,
- $C_1(t_m + T_m + T_q, t_d, T_{r_{i+1}}, T_{c_{i+1}})$ is the AoI area for $t \in [t_m + T_m + T_q, t_d]$, during which the DT is processed at ES_{i+1} and the PS is in domain D_{i+1}.

The AoI calculations for the other cases are similar. In order to save space, we provide the expressions and omit further explanation.

Case $t_m \leq t_b \leq t_m + T_m$ as shown in Figure 5.3. In this case, T_q is calculated as

$$T_q = T_{r_{i,i+1}} + T_{c_{i+1}} + \underline{T_m} T_{c_{i+1}}. \tag{5.4.7}$$

The AoI area is calculated as

$$\begin{aligned} Area_2(t_s, t_m, t_b, T_{bd}) = \\ C_1(t_s, t_b, T_{r_i}, T_{c_i}) + C_2(h_{11}, h_{22}, w_{21}) + \\ C_1(t_{\alpha_2}, t_{\beta_2}, T_{r_{i,i+1}}, T_{c_i}) + C_2(h_{21}, h_{23}, w_{31}) + \\ + C_3(T_q, h_{23}) + C_1(t_m + T_m + T_q, t_d, T_{r_{i+1}}, T_{c_{i+1}}), \end{aligned} \tag{5.4.8}$$

with

$$t_{\alpha_2} = t_m + T_m,$$

$$
\begin{aligned}
t_{\beta_2} &= \underline{(t_m + T_m)}P + T_{r_{i,i+1}} + T_{c_i}, \\
w_{21} &= T_{r_{i,i+1}} + T_{c_i}, \\
w_{31} &= \underline{T_m} T_{c_{i+1}}, \\
h_{21} &= T_{r_{i,i+1}} + T_{c_i} + P, \\
h_{22} &= T_{r_i} + T_{c_i} + T_{r_{i,i+1}} + T_{c_i}, \\
h_{23} &= T_{r_{i,i+1}} + T_{c_i} + P + \underline{T_m} T_{c_{i+1}}.
\end{aligned}
$$

Case $t_b \leq t_m \leq t_d$: This case occurs in the scenario of Figure 5.4b. T_q is calculated as follows:

$$T_q = T_{r_{i,i+1}} + T_{c_{i+1}} + \underline{T_m} T_{c_{i+1}} \tag{5.4.9}$$

and the AoI area is

$$
\begin{aligned}
Area_3(t_s, t_m, t_b, T_{bd}) = \\
C_1(t_s, t_b, T_{r_i}, T_{c_i}) + C_2(h_{11}, h_{21}, w_{21}) + \\
+ C_1(t_{\alpha_3}, t_{\beta_2}, T_{r_{i,i+1}}, T_{c_i}) + C_2(h_{21}, h_{23}, w_{31}) + \\
+ C_3(T_q, h_{23}) + C_1(t_m + T_m + T_q, t_d, T_{r_{i+1}} + T_{c_{i+1}}),
\end{aligned} \tag{5.4.10}
$$

with

$$t_{\alpha_3} = \underline{t_b} P + T_{r_{i,i+1}} + T_{c_i},$$

Case $t_b \leq t_s$ and $t_m \leq t_d$: This case occurs in the scenario of Figure 5.4c. T_q is

calculated as in (5.4.9), and the AoI area is

$$Area_4(t_s, t_m, t_b, T_{bd}) = C_1(t_s, t_{\beta_2}, T_{r_{i,i+1}}, T_{c_i}) + C_2(h_{21}, h_{23}, w_{31}) + C_3(T_q, h_{23}) + C_1(t_m + T_m + T_q, t_d, T_{r_{i,i+1}}, T_{c_i+1}). \quad (5.4.11)$$

Case $t_d \leq t_m \leq T$: In this case, the PS has reached the migration boundary point x_d in D_{i+1} without migrating, i.e., from point x_d and beyond there is no point in considering migration from domain D_i to domain D_{i+1}, and the migration problem becomes the problem of migrating the DT from ES_i to some server ES_{i+2} in a follow-up domain D_{i+2}. Therefore, this case for migration (from D_i to D_{i+1}) starting time t_m is not possible, and we set the migration cost $Cost(t_s, t_m, t_b, T_{bd}) = \infty$ for completeness.
Case $t_m = \infty$: In this case there is no migration. This is shown in Figure 5.4d, The AoI area is calculated as

$$Area_5(t_s, t_m, t_b, T_{bd}) = C_1(t_s, t_b, T_{r_i}, T_{c_i}) + C_2(h_{11}, h_{22}, w_{21}) + C_1(t_{\alpha_3}, t_d, T_{r_{i,i+1}} + T_{c_i}). \quad (5.4.12)$$

Overall, the migration cost is calculated as

$$Cost(t_s, t_m, t_b, T_{bd}) = \begin{cases} \frac{Area_1(t_s,t_m,t_b,T_{bd})}{t_d - t_s}, & t_s \leq t_m + T_m \leq t_b \\ \frac{Area_2(t_s,t_m,t_b,T_{bd})}{t_d - t_s}, & t_m \leq t_b \leq t_m + T_m \\ \frac{Area_3(t_s,t_m,t_b,T_{bd})}{t_d - t_s}, & t_s \leq t_b \leq t_m \leq t_d \\ \frac{Area_4(t_s,t_m,t_b,T_{bd})}{t_d - t_s}, & t_b \leq t_s \leq t_m \leq t_d \\ \infty, & t_d < t_m \leq T \\ \frac{Area_5(t_s,t_m,t_b,T_{bd})}{t_d - t_s}, & t_m = \infty \end{cases} \tag{5.4.13}$$

The discussion above allows us to define the offline version of our problem, i.e., the version where all parameter values and the PS movement until it reaches point x_d are known ahead of time:

Problem 1 (Offline Problem)**.** *Calculate the DT migration time t_m (or decide to not migrate at all) from domain D_i to domain D_{i+1} that minimizes the average AoI cost as defined by* (5.4.13)*.*

5.4.2 The online problem

In the online version of the problem, the PS movement is not deterministic, and, therefore, we do not know its sequence of (location, speed) states $s_t = (x_t, v_t)$ ahead of time. At any time t we know only its history (sequence) of past states and current state $H_t = \{s_1, s_2, \ldots, s_t\}$. Note that given H_t, it can be deduced whether and when the PS crossed the boundary between D_i and D_{i+1}. The online problem is naturally defined as follows:

Problem 2 (Online Problem)**.** *At any time t before PS reaches point x_d, and given the PS history H_t and that DT migration has not yet been initiated, decide whether to initiate the DT migration at t, i.e., set $t_m := t$.*

Note that Problem 2 is solved if the PS reaches x_d without migration or DT migration is initiated at a time t_m. We assume that the PS eventually reaches location x_d, so Problem 2 has always a solution. Also, since crossing from domain D_{i+1} back to D_i would correspond to a different instance of Problem 2, we assume that during its lifetime there only a single domain crossing, that from D_i to D_{i+1} at a time t_b.

Due to the stochastic nature of the PS movement, the hand-off time $\mathbf{t_b}$ and the time period $\mathcal{T}_{bd}$ the PS spends in D_{i+1} until it reaches location x_d are random variables. We will assume that their distributions are known, e.g., by the PS historical data and past trajectory histories, PSs that followed the same route in the past, etc. Hence, cost calculations for Problem 2 will always be expectations over the possible values of $\mathbf{t_b}, \mathcal{T}_{bd}$.

In order to develop online algorithms that solve Problem 2 at a time t, we split the calculation of the average AoI into two parts: The part corresponding to time period $[t_s, t)$ is deterministic since it belongs to the past, and it is the cost already paid. The part of the cost corresponding to time period $[t, t_d]$ is as yet unknown, but we can compute its expectation for different decisions we make at time t (migrate the DT or not) and different future PS state sequences. For the computation of the first part, we define the cost already paid during time $[t-1, t)$ given history h up to time t:

$$c(t, h) = E_{\mathbf{t}_b, \mathcal{T}_{bd}} \left[\frac{\int_{t-1}^{t} AoI(w) dw}{t_d - t_s} | H_t = h \right] \tag{5.4.14}$$

where the $AoI(\cdot)$ is computed by applying the right scenario of Section 5.4.1 that is consistent with h up to now (t), and the values of $\mathbf{t}_b, \mathcal{T}_{bd}$ we condition on. For the (expected) cost $m_t(h)$ of the second component, we first calculate the expected cost

$m_t(h)$ of migrating now (i.e., at time t) when the current history is h:

$$m_t(h) = \begin{cases} E_{\mathbf{t}_b,\mathcal{T}_{bd}}\left[\frac{\int_t^{t_d} AoI(w)dw}{t_d-t_s}|H_t=h\right], & x_s \le x_t < x_b \\ E_{\mathcal{T}_{bd}}\left[\frac{\int_t^{t_d} AoI(w)dw}{t_d-t_s}|H_t=h\right], & x_b \le x_t < x_d \\ \infty, & x_d \le x_t, \end{cases} \quad (5.4.15)$$

where the $AoI(\cdot)$ is computed by applying the right scenario of Section 5.4.1 that is consistent with h up to now (t), and the values of $\mathbf{t}_b, \mathcal{T}_{bd}$ we condition on. Note that when the PS has already crossed the boundary ($x_b \le x_t$) the crossing time random variable is fixed at a deterministic value $\mathbf{t}_b = t_b$. Our online algorithms will need to decide whether to migrate at time t or not, so we define the expected cost that picks the minimum between these two alternatives:

$$g_t(h) = \min\{m_t(h), E_{\mathbf{t_b},\mathcal{T}_{bd}}[Cost(t_s,\infty,t_b,T_{bd})|H_t=h]\} \quad (5.4.16)$$

where $Cost(t_s,\infty,t_b,T_{bd})$ is the cost of no migration calculated in (5.4.12).

5.5 Online algorithms

In this section we use the results of stopping theory to devise online algorithms for Problem 2. The first algorithm is solving the following optimal stopping problem:

$$\nu_{t_s}(h_{t_s}) = \min_{t_s \le t \le T} E_{H_t}[g_t(H_t) + \sum_{j=t_s}^{t-1} c_{t_s}(j, H_t[1:j])|H_{t_s}=h_{t_s}] \quad (5.5.1)$$

where the expectation is over all possible state histories H_t up to time t and given the PS state history $H_{t_s}=h_{t_s}$ up to time t_s. We denote by $H_t[1:j]$ the state history up

to time j, i.e., $H_t[1:j] = h_{t_s} \cup \{s_{t_s+1}, s_{t_s+2}, \ldots, s_j\}$. We assume that for any time t the distribution of H_t given $H_{t_s} = h_{t_s}$ is known, i.e., probabilities $Pr[H_t = f | H_{t_s} = h_{t_s}]$ are known for any sequence f of t states.

Problem (5.5.1) calculates at time t_s the best (in expectation) future time $t_s \leq t_m^* \leq T$ to decide whether start the DT migration or determine not to migrate at all depending on the minimum option in $g_{t_m^*}(H_{t_m^*})$. An online Dynamic Programming (DP) algorithm that solves (5.5.1) is the following: At time $t_s \leq t \leq T$, the algorithm evaluates recursively $V(t, \hat{h}_t)$, defined by

$$V(t, \hat{h}_t) \quad = \quad \min\{g_t(\hat{h}_t), E_{H_{t+1}}[V(t \ + \ 1, H_{t+1}) | H_t \quad = \quad \hat{h}_t] \ + \ c(t, \hat{h}_t)\} \quad (5.5.2)$$

when $t_s \leq t \leq T-1$, and $V(T, \hat{h}_T) = \infty$ for any $\hat{h}_T$. At any time t and given state history $H_t = \hat{h}_t$ so far, the algorithm checks whether $V(t, \hat{h}_t)$ in (5.5.2) achieves its minimum at $g_t(\hat{h}_t)$. If it doesn't, then it waits for one time-slot, and repeats the check at time $t+1$. Otherwise, it outputs as the solution to (5.5.1) time $t_m := t$ and takes the action (migrate or never migrate) that is determined by what quantity achieves the minimum in definition (5.4.16). By setting $V(T, \hat{h}_T) = \infty$ for any $\hat{h}_T$ (base case), we force the algorithm to make a decision before the end of time horizon T, i.e., $t_m < T$. Note that function $V(t, \hat{h}_t)$ is only defined for state histories $\hat{h}_t$ that are consistent with the known state history $H_{t_s} = h_{t_s}$ in (5.5.1), and that the DP table for V is calculated once at time t_s, and then it is repeatedly used. Because of the last characteristic, we denote this algorithm as *non-adaptive* and describe it in Algorithm 5.

Since the PS state sequence that Algorithm 5 encounters is random, its incurred cost is also random. Standard stopping theory results [65] for finite horizon problems

Algorithm 5 Non-Adaptive Online Algorithm

Input: Time t_s, PS state history $H_{t_s} = h_{t_s}$
Output: Stopping time t_m

$t := t_s$
$h_t := h_{t_s}$
$t_m := \infty$
Compute table V
while $t_m = \infty$ **do**
  $s_t :=$ PS state at time t
  $h_t := h_t \cup s_t$
  if $g_t(h_t) \leq E_{H_{t+1}}[V(t+1, H_{t+1})|H_t = h_t]$ **then**
    $t_m := t$
  else
    $t := t + 1$
  end if
end while

return t_m

(recall that problem (5.5.1) has a finite horizon T) imply that the expected cost of Algorithm 5 over all possible state sequences is equal to (5.5.1), i.e., Algorithm 5 solves problem (5.5.1) and, therefore, is optimal in that sense. The time and memory complexity of Algorithm 5 is dominated by the size of the table of V, which is $O(\sum_{t=1}^{T} |\mathcal{H}_t|)$, where $\mathcal{H}_t$ is the set of possible histories up to time t. For example, if histories obey a Markovian model with S states, the size of the V table is $O(TS)$.

A natural extension of Algorithm 5 is to adapt problem (5.5.1) with every new state that is revealed at every time-slot, i.e., define a sequence of problems $\nu_{t_s}(h_{t_s})$, $\nu_{t_s+1}(h_{t_s+1})$, $\nu_{t_s+2}(h_{t_s+2}), \ldots$ and recalculate function table $V_{t_s+k}(t, \hat{h}_t)$ for each $\nu_{t_s+k}(h_{t_s+k})$. The *adaptive* algorithm will settle on a decision (migrate the DT or decide not to migrate at all, depending on (5.4.16)) at the first time t_m we have $V_{t_m}(t_m, h_{t_m}) = g_{t_m}(h_{t_m})$ from (5.5.2). Again, due to the base case $V_{t_s+k}(T, \hat{h}_T) = \infty$, the algorithm

Algorithm 6 Adaptive Online Algorithm

Input: Time t_s, PS state history $H_{t_s} = h_{t_s}$
Output: Stopping time t_m

```
1: t := t_s
2: h_t := h_{t_s}
3: t_m := ∞
4: while t_m = ∞ do
5:    s_t := PS state at time t
6:    h_t := h_t ∪ s_t
7:    Compute table V_t
8:    if g_t(h_t) ≤ E_{H_{t+1}}[V_t(t+1, H_{t+1})|H_t = h_t] then
9:       t_m := t
10:   else
11:      t := t + 1
12:   end if
13: end while
14:
15: return t_m
```

will decide at a time $t_m < T$. The algorithm is described in Algorithm 6.

Note that while Algorithm 5 computes table V once in line 4, Algorithm 6 recomputes it at every time step in line 7, i.e., it performs many more computations in order to achieve better expected cost. As in Algorithm 5, the table size for V starting at a particular time t_0 is $O(\sum_{t=t_0}^{T} |\mathcal{H}_t|)$, for a total of $O(\sum_{t_0=1}^{T} \sum_{t=t_0}^{T} |\mathcal{H}_t|)$ time and memory needed. For example, if histories obey a Markovian model with S states, the size of the V table is $O(T^2 S)$. Since both algorithms need to compute large DP tables, i.e., use a lot of computation time and memory, we propose a more efficient heuristic, which uses the known distribution of future PS states to compute the expected values of $\mathbf{t_b}$, $\mathcal{T}_{bd}$. Using these expectations as predictions of the values t_b, T_{bd} in (5.4.13), we can compute the time t_m that minimizes $Cost(t_s, t_m, t_b, T_{bd})$. Algorithm 7 describes this Best-In-Expectation heuristic, and takes $O(T)$ time.

Algorithm 7 Best-In-Expectation Heuristic

Input: Time t_s, PS state history $H_{t_s} = h_{t_s}$
Output: Stopping time t_m
1: $t_b := E[\mathbf{t_b}|H_{t_s} = h_{t_s}]$
2: $T_{bd} := E[\mathcal{T}_{bd}|H_{t_s} = h_{t_s}]$
3: $t_m := \arg\min_{t_s \leq t \leq t_b + T_{bd}} Cost(t_s, t, t_b, T_{bd})$
4:
5: **return** t_m

5.6 Simulation Results

In the simulation results section, our objective is to evaluate the effectiveness and optimality of the proposed algorithms by considering real-world traffic scenarios and vehicle dynamics. The simulation was conducted using the SUMO software to model a single-lane highway scenario. SUMO is used to derive the probability distribution of hand-off ($\mathbf{t_b}$) and migration deadline period ($\mathcal{T}_{bd}$) for vehicles traveling along a 200 km long highway over a period of 20 days. The highway is divided into two distinct domains (D_i and D_{i+1}) where the hand-off event occurs at a distance of $x_b = 20$ km from the beginning of the road, while the migration boundary point is located 4 km further than x_b, at $x_d = 24$ km. Vehicles were generated randomly from different locations along the road at intervals ranging from 5 to 10 seconds. The maximum speed for all vehicles was set to 130 km/h. The Krauss car-following model [46] was employed to represent the behavior of vehicles in the simulation. This model allows for the specification of different vehicle types based on desired speed, minimum gap, and maximum acceleration.

The vehicles' available information is simplified by representing it as a Markov chain, therefore each pair of vehicle's location (x_t) and speed (v_t) indicates one state, denoted as $s_t = (x_t, v_t)$. To calculate the transition matrix, we discretize the locations

on the highway into BSs with a coverage area of 500 meters [21]. Additionally, the speed is discretized into intervals of 18 km/h. Given the 24 km stretch from x_s to x_d and vehicle speed ranging from 0 km/h to 130 km/h, the Markov chain comprises 392 states.

5.6.1 Probability Distributions of $\mathbf{t}_b$ and $\mathcal{T}_{bd}$

To determine the PDF of $\mathbf{t}_b$ for a given state $s_t = (x_t, v_t)$, we counted the number of vehicles that passed location x_b, along with their respective t_b values. Subsequently, we divided this number by the total number of the vehicles that passed location x_t with speed v_t. The resulting PDF is denoted as $Pr[\mathbf{t_b} = t_b | s_t = (x_t, v_t)]$, which signifies the probability of a vehicle reaching the hand-off location x_b at time t_b, given the current state $s_t = (x_t, v_t)$.

Similarly, the PDF of $\mathcal{T}_{bd}$ is determined using the following formulation:

$$\begin{aligned} &Pr[\mathcal{T}_{bd} = T_{bd} | H_t = (x_t, v_t)] = \\ &\sum_{v_b \in \mathcal{V}} Pr[(x_b, v_b) | (x_t, v_t)] Pr[\mathcal{T}_{bd} = T_{bd} | (x_b, v_b)] \end{aligned} \tag{5.6.1}$$

where $\mathcal{V}$ is a set of all discrete speed values. To obtain $Pr[(x_b, v_b)|(x_t, v_t)]$, we counted the number of vehicles that passed location x_b at speed v_b, and divided the number by the total number of vehicles present at state (x_t, v_t). The probability $Pr[\mathcal{T}_{bd} = T_{bd}|(x_b, v_b)]$ was obtained by first counting the number of vehicles originating from state (x_b, v_b) and then dividing it by the total number of vehicles that reached location x_d.

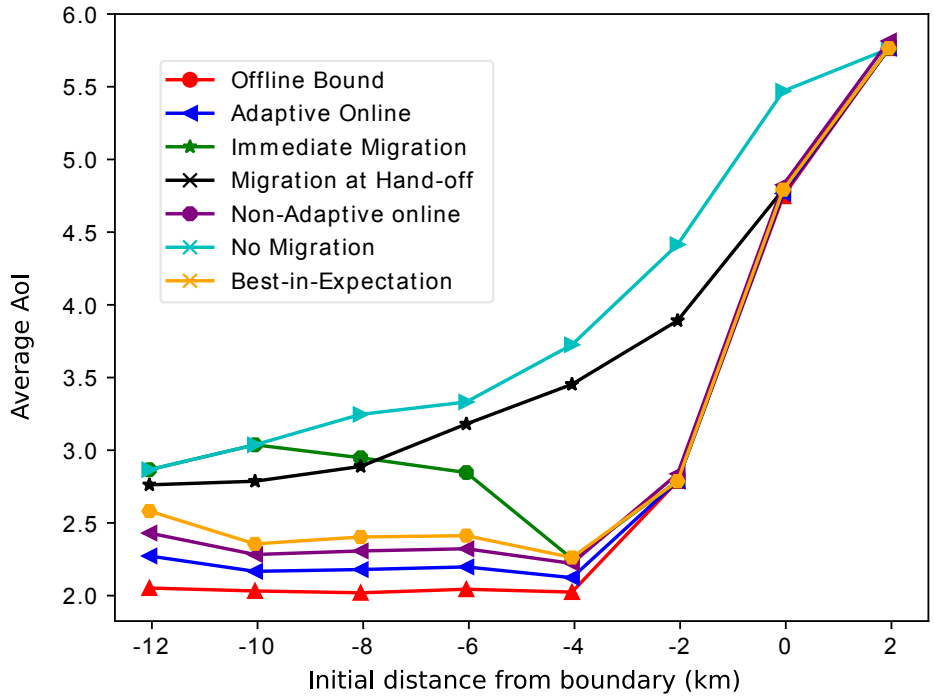


Figure 5.5: Impact of Initial Location Distance from Boundary on Cost of Migration.

5.6.2 Other solutions for comparison

We compare the performance of our proposed online migration solutions with the following solutions:

Offline Bound: This solution assumes complete and precise knowledge of a vehicle's future trajectory and speed at $t = t_s$. The migration initiation time, t_m, is determined by solving the following optimization problem:

$$t_m^* = \underset{t_s \le t_m \le T}{\arg\min}\, \text{cost}(t_s, t_m, t_b, T_{bd}),$$

where $\text{cost}(t_s, t_m, t_b, T_{bd})$ is given in (5.4.13). If the decision is $t_m^* < T$ the migration starts at $t_m = t_m^*$; otherwise, the decision is to not migrate the DT.

Immediate Migration: At $t = t_s$, the expected costs of migration are calculated by assuming $t_m = t_s$ (i.e., migration immediately) and $t_m = \infty$ (i.e., no migration). If the former is smaller, the migration starts at t_s; otherwise, the decision is to not migrate the DT.

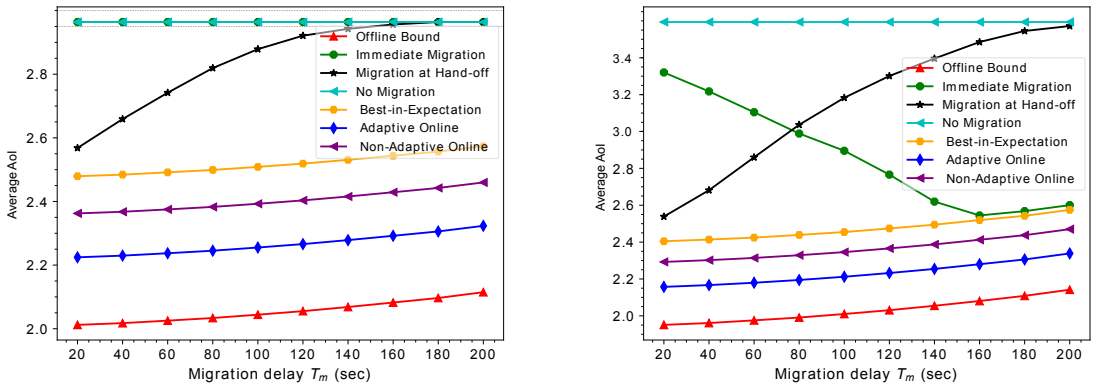
Migration at Hand-off: At $t = t_b$, the expected costs of migration are calculated by assuming $t_m = t_b$ (i.e., migration at hand-off) and $t_m = \infty$ (i.e., no migration). If the former is smaller, the migration starts at t_b; otherwise, the decision is to not migrate the DT.

Best-In-Expectation: as described in Section 5.5.

5.6.3 Simulation results

We compare the average AoI of the proposed solutions with the solutions in the previous subsection by varying the migration delay, the computation delay, the inter-domain transmission delay, the transmission delay in the current and new domains, and the vehicle's initial distance from the domain boundary. We randomly generated 300 vehicles based on the transition matrix obtained from SUMO. Each time-slot in the simulation is 1 second. For the Adaptive and Non-Adaptive Online algorithms, each migration decision-making update happens every 15 time-slots. Other default parameters are listed in Table 5.1. The assumed values for transmission delays (T_{r_1}, T_{r_2}, $T_{r_{12}}$) are based on transmission speeds ranging from 1 Gbps to 5 Gbps for edges between PSs and BSs [80], and 5 Gbps to 10 Gbps for communication between ESs, supported by a 5G network backbone [6]. The computation delays (T_{c_1} and T_{c_2}) are calculated based on the computational capabilities of ESs and the computation requirements of each PS update. Specifically, the computation capabilities of ESs

are uniformly distributed between 10 GHz and 20 GHz. Each vehicle is assigned a computation task with an input data size uniformly distributed between 1 Gbits and 4Gbits, requiring 1 GHz of computation resources per Gbits. Note that the time-average AoI is computed using the discrete event simulation library SimPy [4]. Data from these simulations indicate that our calculated values closely follow the actual values observed, affirming that our approximations effectively capture the real-world dynamics.



(a) x_s is 12 km away from domain boundary

(b) x_s is 5 km away from domain boundary

Figure 5.6: Total cost of migration versus migration delay (T_m)

Figure 5.6 shows the Average AoI as T_m changes over ranges of delays that are consistent with live virtual machine migration [53]. This variability is crucial for modeling the dynamics of DT migrations across different network conditions and server capacities. The "No Migration" cost does not depend on T_m and is the highest among all the methods. The Offline Bound achieves the lowest migration cost due to the knowledge of accurate future information of the vehicle speeds. Among all the heuristic methods, the Adaptive Online algorithm achieves the migration cost that is closest to the Offline Bound. The Non-Adaptive Online algorithm achieves

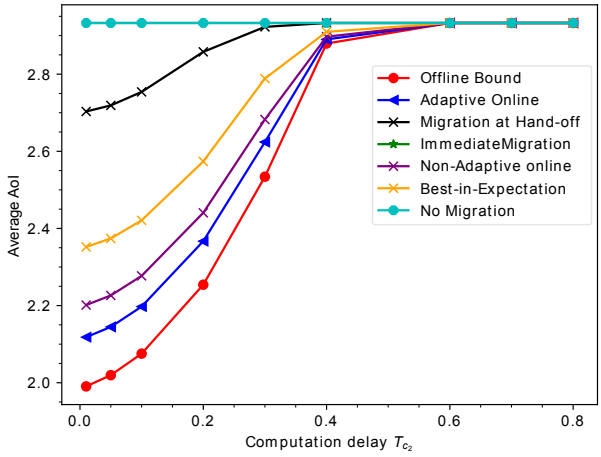
slightly higher migration cost than the Adaptive Online, and the Best-in-Expectation achieves slightly higher migration cost than the Non-Adaptive Online algorithm. For the Immediate Migration heuristic, the decision is to not migration when the initial location (x_s) is 12 km away from the domain boundary and migrate immediately at t_s when x_s is 5 km away from the domain boundary. Since 12 km is a relatively long distance to the boundary, migrating at t_s will result in many DT updates that require cross-domain transmissions when the DT is located in the new ES and the PS is still in the old domain. On the other hand, when x_s is 5 km away from the domain boundary, migrating immediately results in a lower cost than no migration; and furthermore, the migration cost depends on the T_m values, with lower values of T_m leading to faster completion of migration. This rapid migration increases the duration of DT experiencing Inter-domain communication delay, which incurs additional migration cost. For the other methods, increasing T_m in general leads to a corresponding rise in the delay experienced during migration and an accumulation of DT updates, potentially escalating the overall migration cost.

Table 5.1: DT migration simulation default parameters

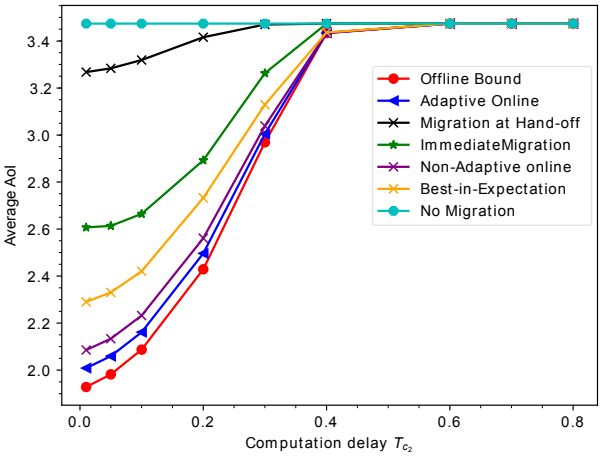
Parameter	Parameter Name	Value
P	Update Period	2 sec
T_{r_1}	Communication delay	800ms
T_{c_1}	Computation delay at ES_1	300 ms
T_{r_2}	Communication delay	500 ms
T_{c_2}	Computation delay at ES_2	100 ms
$T_{r_{12}}$	Inter-domain communication delay	4 sec
T_m	Migration delay	150 sec
t_s	Starting time	0

Figure 5.7 shows the total cost of migration versus the computation delay at

ES_2 (T_{C_2}). Based our parameters, $T_{r_2} < T_{r_1}$, which means hosting the DT in ES_2 requires a lower data transmission delay than in ES_1, if the PS and the DT are in the same domain. However, the decision on the migration time also depends on other parameters. Figure 5.7 shows that when T_{c_2} is sufficiently small, the decision for all the methods (except No Migration) is to migrate, which results in a lower migration cost than No Migration; and when T_{c_2} is sufficiently large, the decision for all the methods is to not migrate. As T_{c_2} increases, the migration costs for all the methods (except No Migration) increases due to the increase in AoI when the DT is hosted at ES_2.



(a) x_s is 12 km away from domain boundary

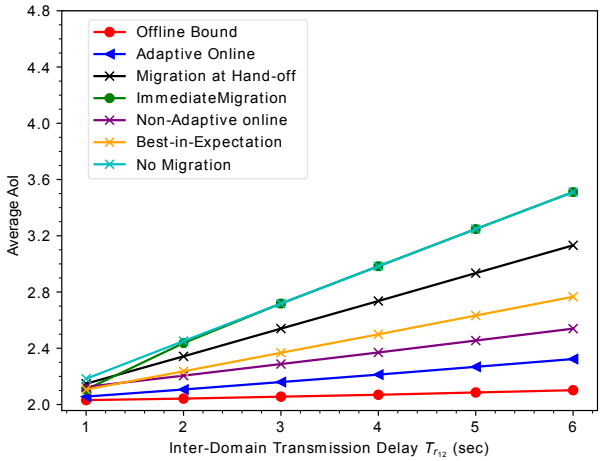


(b) x_s is 5 km away from domain boundary

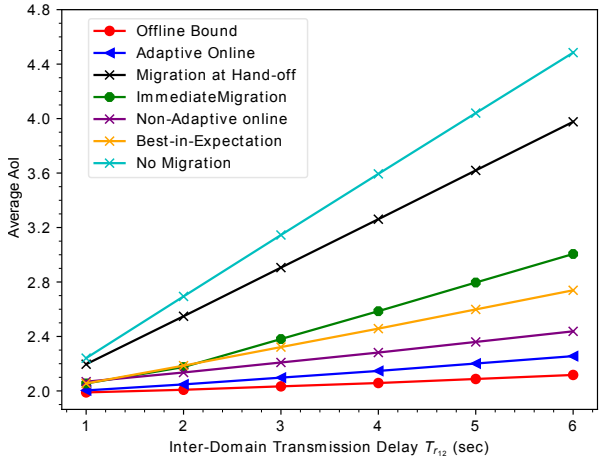
Figure 5.7: Total cost of migration versus computation delay at ES_2 (T_{c_2})

A key observation is that as the initial location of the PS is closer to the boundary—from 12 km to 5 km away—the migration costs become closer to the Offline Bound. It worth noting that, the effectiveness of the Immediate Migration heuristic is more noticeable, where it diverges from No Migration cost and become closer to Offline Bound. Both figures also exhibit a quadratic rise in the cost of migration with

changes in T_{c_2}. This trend is caused by the direct quadratic relationship between the average AoI and the computation delay at ES_2 , as detailed in Section 5.4.1. This quadratic correlation emphasizes the impact of computation delays on the overall migration cost, reinforcing the importance of efficient computational resource allocation strategies in DT enabled vehicular systems.



(a) x_s is 12 km away from domain boundary

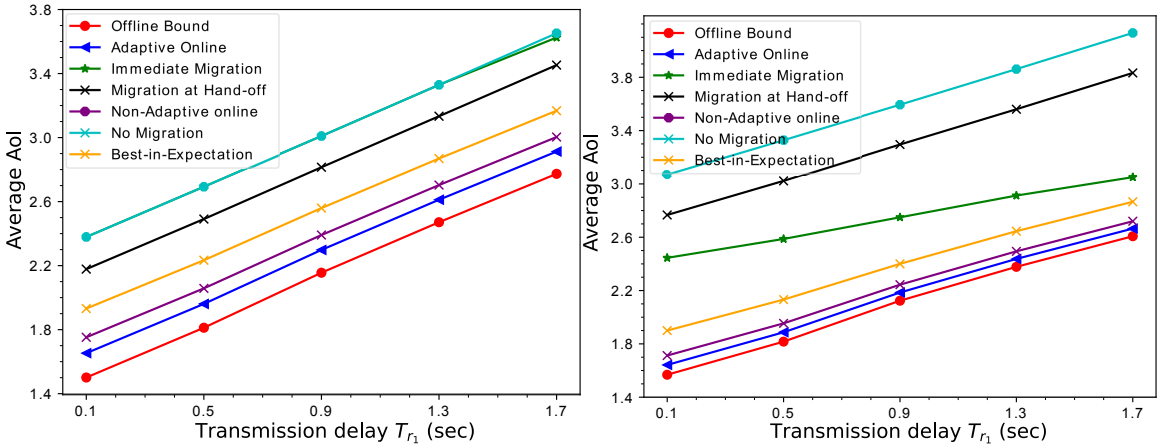


(b) x_s is 5 km away from domain boundary

Figure 5.8: Total cost of migration versus inter-domain transmission delay ($T_{r_{12}}$)

Figure 5.8 shows the total cost of migration versus the cross-domain transmission delay. As $T_{r_{12}}$ increases, the migration cost increases for all the methods. This is because cross-domain transmission is required, regardless of the DT migration method. In addition, the migration cost of the Adaptive Online method is very close to the Offline Bound, and both the Adaptive Online and Non-Adaptive Online algorithm achieve lower migration costs than the other heuristic methods.

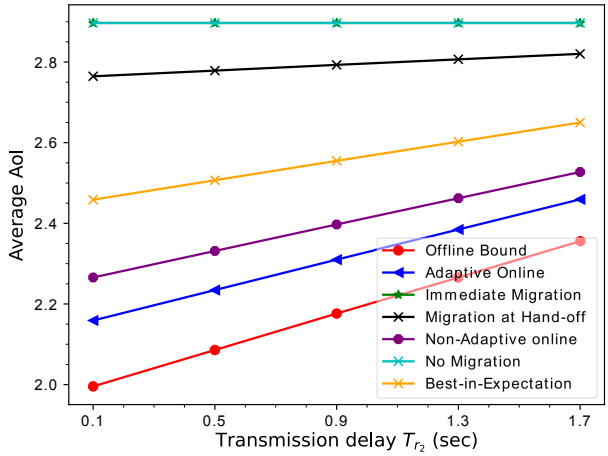
Figures 5.9 and 5.10 show the total cost of migration versus T_{r_1} and T_{r_2}, respectively. In general, the migration cost increases with both T_{r_1} and T_{r_2} for all the methods, since increasing these values increases the AoI at the DT. By comparing the different methods, we have consistent observations as obtained from the previous figures. That is, the Offline Bound achieves the lowest migration cost, the No Migration has the highest migration cost, and the Adaptive Online, the Non-Adaptive Online, the Best-in-Expectation, and the Migration at Hand-off each achieves lower migration cost than the next. The Immediate Migration decides to not migrate when the initial location is 12 km away from the boundary and therefore results in the same migration cost as the No Migration; and it decides to migration at t_s when x_s is 5 km away from the boundary and therefore achieves a migration cost lower than Migration at Hand-off but higher than Best-in-Expectation. A key observation is that as the initial location of the PS is closer to the boundary—from 12 km to 5 km away—the migration costs become closer to the Offline Bound.



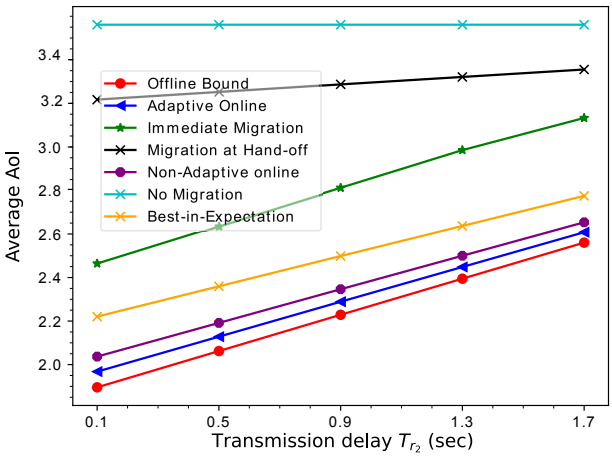
(a) x_s is 12 km away from domain boundary (b) x_s is 5 km away from domain boundary

Figure 5.9: Total cost of migration vs transmission delay in D_1 (T_{r_1})

Both figures also exhibit a linear rise in the cost of migration with changes in T_{r_i}. This trend is caused by the direct linear relationship between the average AoI and the transmission delay, as detailed in Section 5.4.1. This linear correlation emphasizes the impact of transmission delays on the overall migration cost, reinforcing the importance of efficient transmission strategies in DT enabled vehicular systems.



(a) x_s is 12 km away from domain boundary



(b) x_s is 5 km away from domain boundary

Figure 5.10: Total cost of migration vs transmission delay in D_2 (T_{r_2})

Figure 5.5 shows the total cost of migration versus the distance of the initial location of the PS to domain the boundary (x_s), where $x_s = 0$ means that the PS is in the domain boundary and negative x_s means that the PS is in domain D_1. For the Offline Bound, Adaptive Online, Non-Adaptive Online, and Best-in-Expectation, the migration costs decreases slightly as the initial position moves from -12 km away to

$-4\ km$ away from the boundary, where the migration cost reaches the minimum value and then keeps increasing as the initial position moves to and passes the boundary. For the Immediate Migration, the migration cost also reaches the minimum value when the initial position is $-4\ km$ away from the boundary; however, the migration cost initially increases as x_s shifts from -12 km to -10 km, and then it decreases. Since the total migration costs at these locations exceed the cost of No Migration, the decision to Immediate Migration is less favorable and the decision is to not migrate, as detailed in Section 5.6.2. For both the No Migration and Migration at Hand-off, the migration cost increases monotonically as x_s increases.

In summary, the Offline Bound consistently achieves the lowest average AoI, with the Adaptive Online achieving the lowest average AoI among all heuristic solutions, followed by the Non-Adaptive Online and then the Best-in-Expectation. Furthermore, the differences among these solutions become more obvious when the communication and computation delays in the new domain are smaller or the inter-domain transmission delay is larger.

5.7 Summary

This chapter addressed Digital Twin (DT) migration in vehicular systems, focusing on optimizing the Age of Information (AoI) which is crucial for maintaining the relevance and accuracy of the DT in representing its corresponding physical system (PS). We formulated the migration problem as a Markovian stopping problem and proposed an optimal online algorithm using dynamic programming based on vehicular motion statistics. This approach significantly improved DT migration efficiency by minimizing the time-averaged AoI. Moreover, we introduced an adaptive version

of the algorithm, which, although more computationally intensive due to the continuous recomputation of dynamic programming tables at each timestep, further refined the migration decisions based on real-time data. Alongside, a best-in-expectation algorithm was proposed, offering a trade-off between computational efficiency and AoI performance, thereby providing a viable alternative where resources are limited. Comparative analysis with heuristic approaches, such as immediate migration and migration at handoff, emphasized the advantage of the proposed algorithms. Furthermore, an offline algorithm was developed, serving as a benchmark by providing the lower bound on the achievable average AoI. Performance results show that the proposed algorithms can significantly reduce the average AoI compared to heuristic approaches, particularly in high-mobility vehicular environments. The adaptive version of the online algorithm, despite its computational intensity, achieves near-optimal performance, closely approximating the results of the offline algorithm under varying traffic conditions. This demonstrates the effectiveness of dynamic programming when combined with real-time traffic data in managing DT migration efficiently. Overall, the advantage of the proposed algorithms has been demonstrated over a wide range of parameter values.

Chapter 6

Conclusion

6.1 Conclusions

This thesis has explored the multifaceted challenges and solutions associated with deploying Digital Twins (DTs) in wireless communication networks. Through comprehensive analyses and algorithmic developments across three critical chapters, we have significantly advanced the understanding and optimization of DT placement and migration in dynamic network environments.

In Chapter 3, we tackled the complex problem of optimal digital twin (DT) placement in a networked system comprising multiple physical systems (PSs), execution servers (ESs), and an application server (AS). The primary objective was to minimize the maximum data request response delay experienced by the application while ensuring that data age targets are met, reflecting the real-time synchronization requirements inherent in DT applications. We began by formulating the DT placement problem as an integer quadratic program (IQP), incorporating constraints that capture the synchronization requirements between PSs and their corresponding DTs, as

well as the latency constraints between the DTs and the application server. Recognizing the NP-completeness of this problem, we transformed the IQP into a semidefinite program (SDP) relaxation to make it tractable for larger systems. Given the computational complexity, exact polynomial-time solutions are unattainable for practical system sizes. To address this, we developed several polynomial-time approximation algorithms to obtain feasible DT placements. These algorithms—Random Selection, X-Congestion, Z-Congestion, Constraint Slack SDP, and Constraint Slack QP—are based on rounding the fractional solutions of the relaxed SDP to integral assignments. Each algorithm offers different performance trade-offs between minimizing application response delay and adhering to the data age targets. Through extensive simulations, we evaluated the performance of these algorithms under various network configurations and parameter settings. The results demonstrated that the Z-Congestion algorithm consistently outperformed the others in achieving lower constraint violations and better balancing between server load and characteristics. By effectively leveraging congestion information in the Z variables during the rounding process, Z-Congestion led to more balanced assignments of DTs to ESs and improved overall system performance.

Building on the optimization challenges of DT placement explored in Chapter 3, Chapter 4 delves deeper into the the DT placement problem using dynamic flow networks, aiming to minimize the synchronization period between PSs and their corresponding DTs. By viewing DT update data that need to be transmitted and processed as entities to be evacuated, we reduce the DT placement problem to a well-known evacuation problem in operations research. Specifically, we formulate it as a multi-commodity quickest flow problem with flow-dependent transit times, where each

PS-DT pair represents a commodity. Our network model consists of sources (PSs), terminals (DTs), base stations (BSs), edge servers (ESs), and intermediate nodes, all connected through transmission and computational links. The objective is to determine the optimal placement of DTs across available ESs to achieve the minimum synchronization delay while ensuring that all DTs are updated within their required time frames. To solve the problem, we construct a time-expanded network that transforms the dynamic network into a static one over the synchronization period. This allows us to establish a linear programming (LP) formulation for the multi-commodity quickest flow problem. However, the LP solution yields a splittable flow, meaning the data for a single PS-DT pair might be processed across multiple ESs. Recognizing that practical implementations require data flows to be unsplittable—each DT's update data should be processed at a single ES—we proposed an unsplittable flow algorithm to adjust the splittable flows obtained from the LP solution. Additionally, we extended the algorithm to ensure single-path unsplittable flows from each PS to its corresponding DT. Simulation results demonstrated that the unsplittable flow and single-path unsplittable flow solutions perform closely to the optimal solution of the problem, showing good performance in minimizing synchronization period over all PS-DT pairs.

Chapter 5 investigates the Age of Information (AoI) in vehicular DT migration, focusing on optimizing the migration process to maintain low AoI in highly mobile environments. The high mobility of vehicles poses challenges for maintaining low AoI due to varying communication delays and the need for frequent DT migrations to servers closer to the PS. We formulate the problem of determining the optimal time to initiate DT migration as an optimal stopping problem within a Markovian

framework, considering the stochastic nature of vehicular movement. Our objective is to minimize the time-averaged AoI at the DT by deciding when to migrate the DT to a new ES as the vehicle moves across cellular domains. We model the PS-DT synchronization process, data transmission delays, computation delays at the ES, and the migration delay required to transfer the DT to a new ES. To solve the optimal stopping problem, we employ dynamic programming and propose an online algorithm that uses vehicular motion statistics to make migration decisions. We also introduce an adaptive version of this algorithm, which recomputes the dynamic programming tables at each time step to refine the migration decisions based on real-time data. Additionally, we present a best-in-expectation algorithm that offers a sub-optimal but computationally efficient solution by predicting future vehicle states based on expected values. We compare these algorithms with heuristic methods, such as immediate migration and migration at handoff, as well as an offline algorithm that provides a theoretical lower bound on the achievable average AoI. Through simulation results using realistic vehicular traffic models, we demonstrate that the proposed algorithms significantly reduce the average AoI compared to heuristic approaches. The adaptive online algorithm, despite its higher computational complexity, achieves near-optimal performance, closely approximating the results of the offline algorithm under varying traffic conditions. This highlights the effectiveness of our approach in balancing computational efficiency and AoI optimization.

6.2 Future Directions

While this thesis has made significant steps in optimizing DT placement and migration in wireless communication networks, several avenues remain open for future research.

Building upon the foundations established in Chapters 3, 4, and 5, we identify key areas that can further enhance the performance and applicability of DTs in dynamic and complex network environments.

One significant future direction for this thesis is to consider large-scale vehicular networks. In Chapter 5, we assumed that migration decisions are made for a single PS with sufficient resources to accommodate the DT on the new ES. However, in high-density vehicular environments, such as urban areas during peak traffic hours, resource management becomes a critical challenge that must be integrated with migration management. Developing distributed migration algorithms that operate efficiently without centralized control can reduce overhead and enhance scalability.

To further enhance DT-enabled systems, integrating adaptive strategies for DT placement and migration within dynamic network environments is essential. As PSs become increasingly mobile and their movement patterns more unpredictable, particularly in urban vehicular networks, the need for sophisticated trajectory prediction becomes crucial. Traditional mobility models often fail to capture the complexities of real-world PS movements, leading to sub-optimal decisions regarding where DTs should be placed and when they should be migrated. By incorporating advanced trajectory prediction techniques, such as machine learning models trained on historical mobility data obtained from and processed within DTs, we can improve the accuracy of PS movement forecasts. Enhanced predictions enable the network to better anticipate future PS locations, allowing for proactive adjustments in DT placements and preparations for necessary migrations. This proactive approach can significantly reduce synchronization delays and improve overall system performance.

Finally, minimizing power consumption in PSs while ensuring that DT update

periods are minimized is an essential consideration for sustainable network operations. Future work can explore energy-efficient DT management strategies that balance the trade-off between synchronization frequency and power usage. By optimizing communication protocols, data compression techniques, and update schedules, the system can reduce energy consumption without compromising the timeliness of DT updates. This focus on energy efficiency is increasingly important in the context of battery-powered or energy-constrained PSs.

Bibliography

[1] Van Hoof, B. announcing azure digital twins: Create digital replicas of spaces and infrastructure using cloud, AI and IoT. https://azure.microsoft.com/en-us/blog/announcing-azure-digital-twins-create-digital-replicas-\of-spaces-and-infrastructure-using-cloud-ai-and-iot. Accessed: 2018-9-24.

[2] Stockschläger, T. dassault systèmes builds a second singapore. https://www.hannovermesse.de/en/news/news-articles/dassault-systemes-builds-a-second-singapore. Accessed: 2018-3-2.

[3] Deterministic networking. https://datatracker.ietf.org/wg/detnet/about/.

[4] Simpy: Discrete-event simulation for python. https://simpy.readthedocs.io. Accessed: 2023-11-12.

[5] Smartcitiesworld. digital twin created for new indian smart city. https://www.smartcitiesworld.net/news/news/digitaltwin-created-for-new-indian-smart-city-3674. Accessed: 2023-07-23.

[6] 5g backbone. https://www.etsi.org/technologies/5G.

[7] Digital twin. https://www.plm.automation.siemens.com/global/zh/our-story/glossary/digital-twin/24465, 2020.

[8] The digital twin: Compressing time to value for digital industrial companies. https://www.ge.com/digital/applications/digital-twin, 2020.

[9] W. A. Ali, M. P. Fanti, M. Roccotelli, and L. Ranieri. A review of digital twin technology for electric and autonomous vehicles. *Applied Sciences*, 13(10):5871, 2023.

[10] B. R. Barricelli, E. Casiraghi, and D. Fogli. A survey on digital twin: Definitions, characteristics, applications, and design implications. *IEEE Access*, 7: 167653–167671, 2019. doi: 10.1109/ACCESS.2019.2953499.

[11] V. Bayram. Optimization models for large scale network evacuation planning and management: A literature review. *Surveys in Operations Research and Management Science*, 21(2):63–84, 2016.

[12] S. Boschert, C. Heinrich, and R. Rosen. Next generation digital twin. In *Twelfth International Symposium on Tools and Methods of Competitive Engineering (TMCE)*, May 2018.

[13] M. Carey and E. Subrahmanian. An approach to modelling time-varying flows on congested networks. *Transportation Research Part B: Methodological*, 34(3): 157–183, 2000.

[14] H. Chen, T. D. Todd, D. Zhao, and G. Karakostas. Digital twin model selection

for feature accuracy. *IEEE Internet of Things Journal*, pages 1–1, 2023. doi: 10.1109/JIOT.2023.3330411.

[15] H. Chen, T. D. Todd, D. Zhao, and G. Karakostas. Digital twin model selection for feature accuracy in wireless edge networks. In *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–6, 2023. doi: 10.1109/PIMRC56721.2023.10293837.

[16] S. Chen, W. Meng, W. Xu, Z. Liu, J. Liu, and F. Wu. A Warehouse Management System with UAV Based on Digital Twin and 5G Technologies. *International Conference on Information, Cybernetics, and Computational Social Systems*, 2020.

[17] Z. Chen, W. Yi, A. Nallanathan, and J. Chambers. Distributed digital twin migration in multi-tier computing systems. *IEEE Journal of Selected Topics in Signal Processing*, pages 1–15, 2024. doi: 10.1109/JSTSP.2024.3359009.

[18] O. Chukhno, N. Chukhno, G. Araniti, C. Campolo, A. Iera, and A. Molinaro. Optimal placement of social digital twins in edge iot networks. *Sensors*, 20(21): 6181, 2020.

[19] O. Chukhno, N. Chukhno, G. Araniti, C. Campolo, A. Iera, and A. Molinaro. Placement of social digital twins at the edge for beyond 5g iot networks. *IEEE Internet of Things Journal*, 9(23):23927–23940, 2022. doi: 10.1109/JIOT.2022.3190737.

[20] O. Chukhno, N. Chukhno, G. Araniti, C. Campolo, A. Iera, and A. Molinaro. Learning-powered migration of social digital twins at the network edge.

Computer Communications, 226-227:107918, 2024. ISSN 0140-3664. doi: https://doi.org/10.1016/j.comcom.2024.07.019. URL `https://www.sciencedirect.com/science/article/pii/S0140366424002573`.

[21] Y. Dai and Y. Zhang. Adaptive digital twin for vehicular edge computing and networks. *Journal of Communications and Information Networks*, 7(1):48–59, 2022. doi: 10.23919/JCIN.2022.9745481.

[22] 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*, 2020.

[23] J. Deng, Q. Zheng, G. Liu, J. Bai, K. Tian, C. Sun, Y. Yan, and Y. Liu. A Digital Twin Approach for Self-optimization of Mobile Networks. *IEEE WCNC'21*, 2021.

[24] J. Deng, L. Yue, H. Yang, and G. Liu. A digital twin network approach for 6g wireless network autonomy. In *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 415–420, 2023. doi: 10.1109/ICCWorkshops57953.2023.10283539.

[25] Y. Dinitz, N. Garg, and M. X. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, 19(1):17–41, 1999.

[26] Y. Dinitz, N. Garg, and M. X. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, (19):17–41, 1999.

[27] H. Du, S. Leng, J. He, and L. Zhou. Digital twin based trajectory prediction for platoons of connected intelligent vehicles. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–6, 2021. doi: 10.1109/ICNP52444.2021.9651970.

[28] K. Elsayed. A framework for end-to-end deterministic-delay service provisioning in multiservice packet networks. *IEEE Transactions on Multimedia*, 7(3):563–571, 2005. doi: 10.1109/TMM.2005.846779.

[29] M. Ersan, E. Irmak, and A. M. Colak. Applications, insights and implications of digital twins in smart city management. In *2024 12th International Conference on Smart Grid (icSmartGrid)*, pages 378–383, 2024. doi: 10.1109/icSmartGrid61824.2024.10578291.

[30] M. Ersan, E. Irmak, and A. M. Colak. Applications, insights and implications of digital twins in smart city management. In *12th International Conference on Smart Grid*, pages 378–383, 2024.

[31] B. Fan, Y. Wu, Z. He, Y. Chen, T. Q. Quek, and C.-Z. Xu. Digital twin empowered mobile edge computing for intelligent vehicular lane-changing. *IEEE Network*, 35(6):194–201, 2021. doi: 10.1109/MNET.201.2000768.

[32] D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, USA, 2010. ISBN 0691146675.

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

[34] B. Gao, Z. Zhou, F. Liu, and F. Xu. Winning at the starting line: Joint network selection and service placement for mobile edge computing. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1459–1467, 2019. doi: 10.1109/INFOCOM.2019.8737543.

[35] M. Grieves. *Product Lifecycle Management: Driving the Next Generation of Lean Thinking*. New York, NY, USA: McGraw-Hill, 2006.

[36] M. Grieves. Digital twin: Manufacturing excellence through virtual factory replication. *Digital Twin White Paper*, 03 2015.

[37] J. Gu, Y. Fu, and K. Hung. On intelligent placement decision-making algorithms for wireless digital twin networks via bandit learning. *IEEE Transactions on Vehicular Technology*, 2024.

[38] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL `https://www.gurobi.com`.

[39] A. Hagberg, P. J. Swart, and D. A. Schult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), 2008.

[40] A. Hall, K. Langkau, and M. Skutella. An FPTAS for quickest multicommodity flows with inflow-dependent transit times. *Algorithmica*, 47:299–321, 2007.

[41] Z. Hu, S. Lou, Y. Xing, X. Wang, D. Cao, and C. Lv. Review and perspectives on driver digital twin and its enabling technologies for intelligent vehicles. *IEEE Transactions on Intelligent Vehicles*, 7(3):417–440, 2022. doi: 10.1109/TIV.2022.3195635.

[42] M. Huang, W. Liu, T. Wang, A. Liu, and S. Zhang. A cloud–mec collaborative task offloading scheme with service orchestration. *IEEE Internet of Things Journal*, 7(7):5792–5805, 2020. doi: 10.1109/JIOT.2019.2952767.

[43] Y. Huang, J. Du, Z. Yang, Z. Zhou, L. Zhang, and H. Chen. A survey on trajectory-prediction methods for autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 7(3):652–674, 2022. doi: 10.1109/TIV.2022.3167103.

[44] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 261–276, 2009.

[45] E. Kim, Y. Ryoo, B. Yoon, and T. Cheung. Active control and management system for providing the ultra-low latency service on deterministic networks. In *2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 70–74, 2021. doi: 10.1109/ICUFN49451.2021.9528759.

[46] S. Krauss. Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics. *Ph.D. dissertation, Hauptabteilung Mobilität Systemtechnik, Inst. Transp. Res., Porz*, 1998.

[47] N. P. Kuruvatti, M. A. Habibi, S. Partani, B. Han, A. Fellan, and H. D. Schotten. Empowering 6g communication systems with digital twin technology: A comprehensive survey. *IEEE Access*, 10:112158–112186, 2022. doi: 10.1109/ACCESS.2022.3215493.

[48] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby. On the optimal

placement of web proxies in the internet. In *Proceedings IEEE INFOCOM '99*, pages 1282–1290, 1999.

[49] J. Li, S. Guo, W. Liang, J. Wu, Q. Chen, Z. Xu, W. Xu, and J. Wang. Wait for fresh data? digital twin empowered iot services in edge computing. In *2023 IEEE 20th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, pages 397–405, 2023. doi: 10.1109/MASS58611.2023.00056.

[50] J. Li, S. Guo, W. Liang, J. Wang, Q. Chen, Z. Xu, and W. Xu. Aoi-aware user service satisfaction enhancement in digital twin-empowered edge computing. *IEEE/ACM Transactions on Networking*, 32(2):1677–1690, 2024. doi: 10.1109/TNET.2023.3324704.

[51] 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*, 2021.

[52] Z. Liu, N. Meyendorf, and N. Mrad. The role of data fusion in predictive maintenance using digital twin. In *AIP Conference Proceedings*, volume 1949, page 020023. AIP Publishing LLC, 2018.

[53] W. Lu, X. Meng, and G. Guo. Fast service migration method based on virtual machine technology for mec. *IEEE Internet of Things Journal*, 6(3):4344–4354, 2019. doi: 10.1109/JIOT.2018.2884519.

[54] 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*, 8:2276–2288, 2021.

[55] Y. Lu, S. Maharjan, and Y. Zhang. Adaptive edge association for wireless digital twin networks in 6g. *IEEE Internet of Things Journal*, 8(22):16219–16230, 2021. doi: 10.1109/JIOT.2021.3098508.

[56] X. Luo, J. Wen, J. Kang, J. Nie, Z. Xiong, Y. Zhang, Z. Yang, and S. Xie. Privacy attacks and defenses for digital twin migrations in vehicular metaverses. *IEEE Network*, pages 1–1, 2023. doi: 10.1109/MNET.2023.3317320.

[57] J. Lv, Y. Zhao, X. Wu, Y. Li, and Q. Wang. Formal analysis of tsn scheduler for real-time communications. *IEEE Transactions on Reliability*, 70(3):1286–1294, 2021. doi: 10.1109/TR.2020.3026689.

[58] R. Minerva, G. M. Lee, and N. Crespi. Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models. *Proceedings of the IEEE*, 108:1785–1824, 2020.

[59] A. Mosek. The mosek optimization toolbox for matlab manual, 2015.

[60] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury. Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research. *IEEE Communications Surveys Tutorials*, 21(1):88–145, 2019. doi: 10.1109/COMST.2018.2869350.

[61] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J. J. Ramos-Munoz, and J. M. Lopez-Soler. A Survey on 5G Usage Scenarios and Traffic Models. *IEEE Communications Surveys and Tutorials*, 22(2), 2020.

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

[63] Z. Niu, X. S. Shen, Q. Zhang, and Y. Tang. Survey of radio resource management in 5g heterogeneous networks. 2020.

[64] C. I. Papanagnou. A digital twin model for enhancing performance measurement in assembly lines. In *Digital Twin Technologies and Smart Cities*, pages 53–66. Springer, 2020.

[65] G. Peskir and A. Shiryaev. *Optimal stopping and free-boundary problems*. Springer, 2006.

[66] G. Schrotter and C. Hürzeler. The digital twin of the city of zurich for urban planning. *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 88(1):99–112, 2020.

[67] Y. Shu, Z. Wang, H. Liao, Z. Zhou, N. Nasser, and M. Imran. Age-of-information-aware digital twin assisted resource management for distributed energy scheduling. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 5705–5710, 2022. doi: 10.1109/GLOBECOM48099.2022.10000964.

[68] Q. Song, S. Lei, W. Sun, and Y. Zhang. Adaptive Federated Learning for Digital Twin Driven Industrial Internet of Things. *IEEE Wireless Communications and Networking Conference Workshops*, 2021.

[69] M. Soori, B. Arezoo, and R. Dastres. Digital twin for smart manufacturing, a review. *Sustainable Manufacturing and Service Economics*, 2:100017, 2023. ISSN 2667-3444. doi: https://doi.org/10.1016/j.smse.2023.100017. URL `https://www.sciencedirect.com/science/article/pii/S2667344423000099`.

[70] W. Sun, H. Zhang, R. Wang, and Y. Zhang. Reducing Offloading Latency for Digital Twin Edge Networks in 6G. *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY*, 69, 2020.

[71] W. Sun, P. Wang, N. Xu, G. Wang, and Y. Zhang. Dynamic digital twin and distributed incentives for resource allocation in aerial-assisted internet of vehicles. *IEEE Internet of Things Journal*, pages 1–1, 2021. doi: 10.1109/JIOT.2021.3058213.

[72] L. Tang, Z. Cheng, J. Dai, H. Zhang, and Q. Chen. Joint optimization of vehicular sensing and vehicle digital twins deployment for dt-assisted iovs. *IEEE Transactions on Vehicular Technology*, 73(8):11834–11847, 2024. doi: 10.1109/TVT.2024.3373175.

[73] F. Tao and M. Zhang. Digital twin shop-floor: A new shop-floor paradigm towards smart manufacturing. *IEEE Access*, 5:20418–20427, 2017. doi: 10.1109/ACCESS.2017.2756069.

[74] F. Tao, H. Zhang, A. Liu, and A. Nee. Digital Twin in Industry: State-of-the-Art. *IEEE Transactions on Industrial Informatics*, 15:2405–2415, 2019.

[75] P. Tiwari. Mobility digital twin with connected vehicles and cloud computing. *IEEE Internet of Things*, 2021.

[76] S. Tschöke, F. Lynker, H. Buhr, F. Schreiner, A. Willner, A. Vick, and M. Chemnitz. Time-sensitive networking over metropolitan area networks for remote industrial control. In *2021 IEEE/ACM 25th International Symposium*

on Distributed Simulation and Real Time Applications (DS-RT), pages 1–4, 2021. doi: 10.1109/DS-RT52167.2021.9576141.

[77] 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*, pages 1–14, 2011.

[78] M. Vaezi, K. Noroozi, T. D. 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. doi: 10.1109/JIOT.2022.3200327.

[79] M. Vaezi, K. Noroozi, T. D. 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. doi: 10.1109/JIOT.2022.3200327.

[80] M. Vaezi, K. Noroozi, T. D. Todd, D. Zhao, and G. Karakostas. Digital twin placement for minimum application request delay with data age targets. *IEEE Internet of Things Journal*, 2023.

[81] 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*, pages 468–476. IEEE, 2018.

[82] W. Wang, Y. Yang, L. U. Khan, D. Niyato, Z. Han, and M. Guizani. Digital twin for wireless networks: Security attacks and solutions. *IEEE Wireless Communications*, 31(3):278–285, 2024. doi: 10.1109/MWC.020.2200609.

[83] 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), pages 1–6, 2020. doi: 10.1109/VTC2020-Spring48590.2020.9128938.

[84] G. White, A. Zink, L. Codecá, and S. Clarke. A digital twin smart city for citizen feedback. *Cities*, 110:103064, 2021.

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

[86] Q. Xu, J. Wang, and K. Wu. Learning-based dynamic resource provisioning for network slicing with ensured end-to-end performance bound. *IEEE Transactions on Network Science and Engineering*, 7(1):28–41, 2020. doi: 10.1109/TNSE.2018.2876918.

[87] X. Xu, B. Shen, S. Ding, G. Srivstava, M. Bilal, M. R. Khosravi, V. G. Menon, M. A. Jan, and M. Wang. Service Offloading with Deep Q-Network for Digital Twinning Empowered Internet of Vehicles in Edge Computing. *IEEE Trans. On Industrial Informatics*, 2020.

[88] Z. Yang, M. Chen, Y. Liu, and Z. Zhang. Optimizing synchronization delay for digital twin over wireless networks. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 9106–9110, 2024. doi: 10.1109/ICASSP48485.2024.10446262.

[89] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus. Age of information: An introduction and survey. *IEEE Journal on Selected Areas in Communications*, 39(5):1183–1210, 2021.

[90] Z. Yin, N. Cheng, T. H. Luan, and P. Wang. Physical layer security in cybertwin-enabled integrated satellite-terrestrial vehicle networks. *IEEE Transactions on Vehicular Technology*, 71(5):4561–4572, 2022. doi: 10.1109/TVT.2021.3133574.

[91] J. Yu, A. Alhilal, P. Hui, and D. H. K. Tsang. Bi-directional digital twin and edge computing in the metaverse. *IEEE Internet of Things Magazine*, 7(3): 106–112, 2024. doi: 10.1109/IOTM.001.2300173.

[92] Q. Yu, J. Ren, Y. Fu, Y. Li, and W. Zhang. Cybertwin: An Origin of Next Generation Network Architecture. *IEEE Wireless Communications*, 26:111–117, 2019.

[93] K. Zhang, J. Cao, S. Maharjan, and Y. Zhang. Digital twin empowered content caching in social-aware vehicular edge networks. *IEEE Transactions on Computational Social Systems*, 9(1):239–251, 2022. doi: 10.1109/TCSS.2021.3068369.

[94] Y. Zhang and W. Liang. Cost-aware digital twin migration in mobile edge computing via deep reinforcement learning. In *2024 IFIP Networking Conference (IFIP Networking)*, pages 441–447, 2024. doi: 10.23919/IFIPNetworking62109.2024.10619837.

[95] Y. Zhang, W. Liang, W. Xu, Z. Xu, and X. Jia. Cost minimization of digital twin placements in mobile edge computing. *ACM Trans. Sen. Netw.*, 20(3), May 2024. ISSN 1550-4859. doi: 10.1145/3658449. URL `https://doi.org/10.1145/3658449`.

[96] L. Zhao, Z. Bi, A. Hawbani, K. Yu, Y. Zhang, and M. Guizani. Elite: An intelligent digital twin-based hierarchical routing scheme for softwarized vehicular networks. *IEEE Transactions on Mobile Computing*, 22(9):5231–5247, 2023. doi: 10.1109/TMC.2022.3179254.

[97] L. Zhao, Z. Zhao, E. Zhang, A. Hawbani, A. Y. Al-Dubai, Z. Tan, and A. Hussain. A digital twin-assisted intelligent partial offloading approach for vehicular edge computing. *IEEE Journal on Selected Areas in Communications*, 41(11): 3386–3400, 2023. doi: 10.1109/JSAC.2023.3310062.

[98] Y. Zhong, J. Wen, J. Zhang, J. Kang, Y. Jiang, Y. Zhang, Y. Cheng, and Y. Tong. Blockchain-assisted twin migration for vehicular metaverses: A game theory approach. *Transactions on Emerging Telecommunications Technologies*, page e4856, 2023.

[99] P. Zhou, J. Zhu, Y. Wang, Y. Lu, Z. Wei, H. Shi, Y. Ding, Y. Gao, Q. Huang, Y. Shi, et al. Vetaverse: A survey on the intersection of metaverse, vehicles, and transportation systems. *arXiv preprint arXiv:2210.15109*, 2022.

[100] Y. Zhou, J. Wu, X. Lin, A. K. Bashir, Y. D. Al-Otaibi, and H. Xu. Secure digital twin migration in edge-based autonomous driving system. *IEEE Consumer Electronics Magazine*, 12(6):56–65, 2023. doi: 10.1109/MCE.2022.3217363.