A DIGITAL TWIN SYNCHRONIZATION PROTOCOL FOR BANDWIDTH-LIMITED IOT APPLICATIONS

BY

DEEPTI KALASAPURA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2023

Urbana, Illinois

Adviser:

Professor Matthew Caesar

ABSTRACT

Digital Twins are evolving as a key component in modern systems with diverse applications like remote prognostics, optimizing run-time operation, anomaly detection, and more. The essential elements of a digital twin are a virtual representation, a physical asset, and the transfer of data/information between the two. IoT deployments are generally characterized by resource constraints, making synchronization of digital twins with IoT devices more challenging. There is a pressing need to optimize the bandwidth of the data transferred between the system and the twin while ensuring that the twin is able to capture selected key aspects of the current operational state accurately. This thesis explores the question: is it possible to synchronize twin state with practically low bandwidth requirements while simultaneously satisfying required accuracy constraints? To study this question, we present TwinSync, a framework that can be utilized to construct flexible real-time representations of deployed IoT systems and efficiently synchronize relevant system states with the twin, over a communication bottleneck, within a configurable application-specific notion of error (henceforth referred to as *weak synchronization*). Our approach is optimized to achieve data transfers utilizing less bandwidth without compromising the ability of the twin to replicate real-time system states within the specified approximate synchronization semantics. We evaluate the efficacy of TwinSync's synchronization by conducting both a synthetic analysis and a case study based on a real-life application prototype. Our evaluation indicates that using TwinSync can provide the same or greater accuracy (in many cases) while sending significantly fewer bytes than a bandwidth-insensitive synchronization approach. The result is attributed to a more judicial selection of data to transmit over bottlenecks, compared to bandwidth-insensitive approaches.

To Amma, Anna and Amogh, for their love and support.

ACKNOWLEDGMENTS

My journey at the University of Illinois at Urbana-Champaign would not have been possible without the support of multiple people.

A very big thank you to my advisor, Professor Matthew Caesar, for his continued guidance and support. His careful and considerate feedback encouraged me to push the bounds of my research and implement newer ideas.

I would like to thank Professor Tarek Abdelzaher for his mentorship and all the excellent suggestions he proposed during our meetings. Frequent brainstorming discussions with him helped me come up with new ideas and solidify the technical content of this work. I am truly grateful for the advice and contributions of Jinyang Li without whom this research would not see the light of day. I would also like to thank my collaborators, Shengzhong Liu, Yizhuo Chen, and Ruijie Wang who helped push this work in the right direction.

I would like to thank Dr. Jae Kim, and Dr. Guijun Wang from The Boeing Company for their experienced insights and constant encouragement. I would also like to thank all the supporting organizations that provided funds for this work. This thesis is based on the work that was supported in part by The Boeing Company and the IBM-Illinois Discovery Acceleration Institute (IIDAI) and in part by ARL Cooperative Agreement W911NF-17-2-0196, and NSF CNS 20-38817.

A huge thank you to my family for their unconditional love and unbridled support, I would not be here without them. I have had the privilege of being in the company of great friends and could not have completed this thesis without their support. A special thank you to Rahul, Arpitha, and Chandana for their constant encouragement. Thank you.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

IoT technology has seen a massive uptake across a broad spectrum of industry and social sectors [1]. It is being used to enable applications such as telemedicine [2], pervasive computer vision [3], edge-based natural language processing [4], automation of factories and logistics plants [5], and enhanced supply chain resiliency [6]. Unfortunately, IoT systems face operational challenges – they often rely on remote deployments, which may spread over large regions and use large numbers of complex autonomous devices. Developers and operators are separated from their physical assets making it difficult to observe the system state. These challenges lead to a number of problems, including security issues and vulnerabilities, and difficulty in monitoring, management, configuration, deployment, and testing. Given the nature of these systems, digital twinning methodologies are an ideal solution to many of these problems [7].

The concept of digital twinning has received great recent attention within the IoT research community [7]. The ability to remotely "recreate" selected aspects of the operational processes of a remote device stands to greatly simplify aspects of management, security, and reliability of the system. Towards this end, several proposals have been made to digitally twin IoT devices in various domains such as industrial machinery [8], port operations [9], energy systems [10], and model-based systems engineering [11]. These solutions extend the state of the art by defining digital twinning mechanisms at various levels of the system development life-cycle and approaches to represent and aggregate state such as parameters and measured values across multiple components.

A core mechanism used in these efforts is to automatically and continuously replicate selected aspects of the state of the remote IoT device into a local copy at the twin that can be directly observed and manipulated by human operators. Such replication is referred refer to as *synchronization* of IoT system state with the twin. In general, depending on the application, only selected aspects of system state need to be synchronized. For example, in developing a digital twin for tire health monitoring (for aircraft landing) [12], the key variables to synchronize might be tire temperature, pressure, velocity vector, and age. In IoT applications, the ability to accurately replicate the real-time state of a remote device is challenging. This is so because resources such as bandwidth and power are constrained, yet the precision and accuracy of observations are critical. It remains unclear how the state of the remote device can be effectively twinned while honoring various resource limitations. While valid approaches may include continuous state replication, such techniques may not scale well or may exceed bandwidth constraints between the IoT devices and the twin. On

the other hand, looser-grained replication, such as copying the state at sufficiently large intervals, may miss important changes or impair the understanding of remote behavior, the main purpose of Digital Twinning.

This thesis explores the question: is it possible to synchronize twin state with practically low bandwidth requirements while simultaneously satisfying required accuracy constraints? We present *TwinSync*, a framework for *weakly synchronizing* system states in digital twinning systems, taking into account considerations such as the cost of synchronizing each variable and the importance of that variable to the application. TwinSync directly optimizes a configurable application-specific notion of "permissible error" in key outputs of the digital twin (which we refer to as *weak synchronization*). This error leeway is exploited to significantly reduce synchronization costs. The weak synchronization semantics of TwinSync and the resulting savings in synchronization cost are what distinguish this approach from prior digital twin literature. To address the above challenges, TwinSync adopts a form of *weak synchronization* that is novel in allowing operators to define *an application-specific* notion of error in key twin outputs. We call such specifications, *operator intents*. Operator intents are then backtracked from the specified accuracy requirements on twin outputs to finer-grained constraints on the error in individual synchronized variables using dynamically retrieved models of system behavior. Given operator intents, these models are used to decide the states that have to be synchronized and their acceptable synchronization errors and schedules. Synchronization is then carried out to minimize cost subject to the aforementioned constraints. TwinSync makes use of sensitivity analysis and optimization techniques to automatically derive schedules and synchronization policies for coordinating state updates to the remote twin while observing the stated weak synchronization semantics. The created (approximate) real-time view of application-relevant aspects of the system can then be used on the twin to enable value-added functionality such as optimizing system configuration, anomaly detection, remote troubleshooting, or "what-if" analysis. This thesis focuses on the synchronization protocol and its relations to other digital twin components.

To evaluate TwinSync and build a stronger understanding of the trade-offs involved in synchronizing system state across resource bottlenecks, we experiment with TwinSync's behavior in both synthetic and real-world contexts. Specifically, we first simulate and analyze TwinSync's performance across different behavioral optimization goals (or operator intents) and levels of sensitivity of desired behavioral outputs with respect to system state variables, contrasting it with baseline measures. In order to gather deeper insights, we integrate TwinSync with an IoT system, called IoBT OS [13], designed for target tracking applications, and test its performance in terms of both synchronization overhead and higher-level application metrics, thereby demonstrating the advantages of the new protocol from an application's

perspective.

The rest of this thesis is organized into the following chapters:

- Chapter 2 provides background on how digital twinning concepts can be used to address many challenges of IoT systems and reviews recent work. It also provides an overview of the proposed approach and describes the main contributions of this thesis.
- Chapter 3 describes TwinSync's capabilities and the system architecture.
- Chapter 4 describes our intent-driven weak synchronization protocol.
- Chapter 5 discusses our implementation of TwinSync, an application of it for a particular case study, and evaluates our synchronization approach across multiple dimensions to demonstrate the effectiveness of TwinSync.
- We discuss future extensions and conclude in Chapter 6

CHAPTER 2: RELATED WORK

In this chapter we review background work done in the IoT and digital twinning domains. We first discuss the various applications of IoT systems and their pervasiveness across industries (Section 2.1). We then provide an overview of the digital twinning approaches presented in recent works (Section 2.2) and various state synchronization methods used for digital twinning and other applications (Section 2.3). We conclude by presenting the research opportunities in this space and the main contributions of this thesis (Section 2.4).

2.1 IOT SYSTEMS AND APPLICATIONS

IoT-based solutions have a ubiquitous presence in all realms of society. They are used in smart mobility to optimize transportation modes, provide driving recommendations, and improve road safety [14, 15, 16]. Smart grid technologies leverage IoT to monitor changes in electricity usage. Significant research including routing protocols for wireless sensor networks [17], game theory for consumption schedules [18], and energy-efficiency spectrum discovery schemes [19] has been conducted to enable the utilization of IoT in smart grid construction. Smart homes are also becoming more common with novel approaches to enhance inter-device communications such as a radio-frequency-based smart home controller [20], micro-web servers [21], and Arduino Ethernet shields [22]. IoT technology is also used in public safety and environment monitoring [23], healthcare [24], industry [25], agriculture [26], and numerous other domains. However, there are still many open issues plaguing IoT systems such as availability, reliability, mobility, scalability, resource management, and visualization. [27].

2.2 DIGITAL TWINNING

Digital twins are virtual representations of systems that mimic the behavior of physical assets, enabled by a mechanism for state observation. NASA is commonly credited for advancing early digital twin technology [28], as they introduced one of the earliest applications of digital twins during the Apollo 13 mission, where NASA leveraged high-fidelity representations of the aircraft to simulate the real-time situation and mitigate a problem with the oxygen tanks [29]. Today digital twins are a formal engineering process well integrated into many domains such as automotive [30], military [31], manufacturing [32], medicine [33], city planning [34], and more [35]. Digital twinning technology has enabled a fundamental advance

in building reliable decentralized systems by providing a tool-set for monitoring, diagnostics, and prognostics. Research on twinning mainly focused on constructing application-specific definitions and architectures. Liu *et al.* [36] present a digital twin-driven approach to developing models of human-robot collaborative assembly. Verdouw *et al.* [37] analyze how digital twinning can be used to advance smart farming and presents six variants of digital twins based on their role in the lifecycle of products. Ghosh *et al.* [38] develop two computerized systems capable of constructing and adapting digital twins for machine tools from real-time sensor signals and historical data sets. Sharma *et al.* [39], review the current state of the art and observe a lack of clarity in the definition and generic system architecture of digital twins, they also identify data-sharing techniques as an area with research potential. However, there have been some efforts to standardize the digital twin development process such as Picone *et al.* [40] who propose White Label Digital Twins (WLDT) a Java library that provides a set of core functionalities to mirror and process smart objects. Digital twins of virtual machines are also well explored and can be constructed using live migration techniques. Lei *et al.* [41] propose a novel hybrid-copy algorithm that can overcome the defects of the pre-copy and post-copy algorithms by using a Markov model to forecast the memory access pattern on the Xen platform. Lim *et al.* [36] propose a digital twin classification along four dimensions to reconcile the multitude of implementations proposed in the literature. The first criterion is the mode of data flow between the physical system and the model. Tao *et al.* [42] present a twinning framework for product design and Sivalingam *et al.* [43] describe a methodology for twinning power converters using unidirectional data transfers from the physical system to the twin. On the other hand, some strategies such as twinning CNC (computer numerical control) machine tools [44] and container unloading [45] support bidirectional communication and allow the twin to reconfigure the system. The second aspect of classification is the virtual model type. There is a multitude of representations used such as geometric [42, 46], analytical [47], state-based [48], kinematics [49], and so on. The third dimension is based on the network protocol used for aiding inter-device communications. Commonly used IoT protocols include HTTP [44], MQTT (Message Queuing Telemetry Transport) [50], and CoAP (Constrained Application Protocol) [51]. The fourth property used for classification is the simulation software for running the models.

2.3 STATE SYNCHRONIZATION

A key aspect of digital twinning technology is the synchronization mechanism used to update the twin with information about the state of the real system. Existing digital twin approaches face scaling challenges. IoT systems involve large-scale intricate programs

which are often tightly dependent on the environment. They also operate in bandwidth and resource-constrained environments with the need for strong privacy and security guarantees. Digital twin synchronization strategies generally focus on attaining higher accuracy which directly impacts bandwidth consumption. Akbarian *et al.* [52], present three approaches to synchronize digital twins in industrial settings by using Kalman filters and Proportional, Integral, Derivative (PID) controllers that receive the real system output signal as a reference signal. Han *et al.* [53] propose a solution to synchronize digital twins in the Metaverse through a dynamic hierarchical framework, in which a group of IoT devices in the lower level are incentivized to collectively sense physical objects' status information and virtual service providers in the upper level determine synchronization intensities to maximize their payoffs. Zipper *et al.* [54], present an approach to synchronize digital twins of industrial plants by minimizing errors in the output signals generated by the twin and the system, this optimization is repeated at a fixed frequency. Overall, despite extensive research efforts to ensure the transfer of relevant data, there has not been much work on *weak synchronization* (in the digital twin synchronization space) to address the unique resource constraints of IoT systems. This work aims to bridge this gap by focusing on minimizing bandwidth overhead while offering well-defined synchronization accuracy semantics.

Approximate synchronization encapsulates data transfers where an intelligent policy is used to send information across distributed networks rather than sharing the entirety of data. The digital twin synchronization problem is similar to other work describing efficient data copying techniques or approximate agreement outside of the digital twin context. For example, Beuchert *et al.* [55] propose an event-triggered learning-based approach to reduce bandwidth overhead for signals exhibiting time-varying cyclic patterns. Suh *et al.* [56] present a send-on-delta algorithm with a linear-predictor transmission strategy, where data are sent if the predicted value of the signal differs from the actual current value by a fixed threshold. Mo *et al.* [57] discuss a multi-step sensor selection strategy with the goal of minimizing an objective function related to the Kalman filter error covariance. Battistelli *et al.* [58] estimates state based on data collected from multiple sensors within communication rate limits by relying on the sensor nodes to compute locally optimal estimates. Cheng *et al.* [59] experiment with conventional lossy compression algorithms on inertial motion data in a multi-sensor network and observe that communication overhead can be reduced at a cost of some added latency. While TwinSync can be structured on top of some of the above approaches, this protocol is novel in automatically linking low-level synchronization error objectives to higher-level accuracy guarantees stated directly on operator-specified variables and translated to the low-level objectives using application models on the twin.

2.4 RESEARCH OPPORTUNITIES

Digital twin research so far has mainly focused on the architecture and modeling of virtual representation. The synchronization is mostly delegated to standard communication protocols or coarse-grained rules [44, 50, 51]. The topology of IoT systems is constantly changing and a single deployment can be reconfigured to run a multitude of applications. TwinSync's ability to dynamically query and construct a real-time view of the system removes the rigid dependencies current approaches to digital twinning possess. TwinSync can be particularly useful in IoT scenarios where there is an emphasis on reducing network communication footprint, conserving resources, and accelerating sensing-to-decision loops. In such environments, the twin can generate a global view of the system state, as opposed to the local information individual sensors possess. This provides a surface for running various analytical modules like system optimization, anomaly detection, and so on in addition to the inherent benefits obtained by a digital twin. In this thesis, we aim to efficiently define, construct, and synchronize digital twins while accounting for the unique resource limitations encountered by IoT systems. The main contributions of this work are:

1. An extensible system architecture for easy integration of digital twins and modular value-added functions (Chapter 3).

2. A novel synchronization protocol that accounts for operator intentions to minimize the synchronization bandwidth overhead (Chapter 4).

3. Demonstration of the performance and application of the protocol in a real-time IoT system (Chapter 5).

CHAPTER 3: SYSTEM ARCHITECTURE

In order to study the possibility of synchronizing twin states with practically low bandwidth requirements while simultaneously satisfying required accuracy constraints we must define a set of abstractions and an architecture that can describe the characteristics of the twin state and how it is computed. In this chapter, we first discuss an overview of what our architecture aims to accomplish (Section 3.1). We then define the constructs that guide the twin construction and synchronization (Section 3.2), describes the digital twin hypervisor (Section 3.3), and present the layers of the system architecture (Section 3.4).

3.1 OVERVIEW

The TwinSync weak synchronization protocol is designed to function within an extensible framework that can be used to digitally twin a broad range of IoT systems operating in resource-constrained environments. It is built on a set of abstractions that can be generalized to various IoT systems to generate their digital twins. Our broad goal is to provide an end-to-end solution to efficiently define, synchronize and update the digital twin with the physical asset. However, we also recognize that the global system state view captured by the digital twin provides a rich source of information that can transform the digital twin from a passive observer into an active agent. The synchronization protocol and value-added functions such as the system optimizer module establish a feedback loop between the physical system and the twin. If we shift our perspective and view the twin as a form of input for additional computation and analysis we can define a library of such value-added functions. These functions can provide additional insights to the operator throughout the life-cycle of the system. In the design phase, different configurations can be simulated and validated against a set of safety properties. During run-time, anomaly detection, root cause analysis, system re-configuration, and other analytical modules can be executed.

Different applications might have different objectives from twinning that correspond to different twin outputs. As such, they may need different subsets of state to be synchronized and/or may impose different requirements on individual state variable synchronization errors. We thus consider applications where operators are concerned about some specific (operator-defined) set of system characteristics, such as, say, reducing target detection latency, improving localization accuracy, or having an acceptable activity recognition precision/recall. As mentioned in the introduction, we call such higher-level behavioral or performance goals, operator *intents*. High-level models describe the dependency of these performance charac-

teristics on various environmental conditions and state variables in different system configurations. TwinSync is thus an intent-driven protocol that derives its synchronization policies from operator intents while meeting high-level application-specific accuracy objectives. As common to other digital twinning frameworks, the digital twin in our architecture would typically be used to perform a range of modular value-added functions, such as anomaly detection, "what if" analysis, configuration optimization, or troubleshooting. The execution of these functions requires the twin to know the current remote device state pertinent to the declared intents. For example, if one intent is to minimize end-to-end latency for a workflow involving multiple IoT devices, then the current network conditions among these devices would constitute the relevant state.

Below, we first discuss the overall architecture and explain the mechanisms by which high-level intents are translated into actionable decisions for the weak synchronization protocol. Chapter 4 details the actual protocol and derives the synchronization assurances it offers.

3.2 PROBLEM DEFINITION

In order to arrive at an optimal synchronization schedule between the IoT system and the twin, TwinSync must know three pieces of information: (i) what components are used in the IoT system being twinned, (ii) what their relevant behavioral models are, and (iii) what the operator intents are. Below, we elaborate on those three inputs in more detail.

The component library: The twinned IoT system is composed of components that can be selected from a pre-defined library. The IoT application can therefore be thought of as a set of used components, $\mathcal{C}$, where each component $c \in \mathcal{C}$ has typed inputs, I_c, and typed outputs, O_c. An interconnection topology, $G(V, E)$, describes how component inputs and outputs are connected in the current application workflow. Specifically, the set of nodes, V, represents the set of component inputs and outputs in use, whereas the set of edges, E, describes their interconnections in the application workflow. Each component is also associated with a set of parameters, $x_{c,i}$, that we divide into *configurable* inputs ($x_{c,i}^{config} \in I_c$), *locally changing* outputs ($x_{c,i}^{out} \in O_c$) and *fixed* constants, $x_{c,i}^{fixed}$, for a particular application. (While the values of some of these parameters change over time, in this chapter, we omit their time index for notational simplicity, where no ambiguity arises.)

Behavioral component models: Each component, besides its actual implementation and its input/output specification, has a set of models, $\mathcal{M}_c$, in the library that abstracts its behavior. Each model, $M_c \in \mathcal{M}_c$, focuses on a different aspect of component behavior. For example, a target detection algorithm can have a model for computational latency (as a function of the underlying hardware platform), and a model for target recognition accu-

racy (as a function of various environmental conditions, hyperparameters, and configuration options). Those models estimate relevant characteristics of component behavior, which we henceforth call *behavioral outputs*. Each component $c \in \mathcal{C}$ is thus associated with a set of behavioral outputs, B_c. Unlike outputs O_c, described above, behavioral outputs are not values computed by the component itself. Rather, they represent outputs of models, M_c, that estimate various behavioral characteristics of the component, such as its latency, quality, bandwidth consumption, or probability of failure (as computed by the model). Component models can be hierarchically composed into models of higher-level subsystems, ultimately producing the overall workflow model. In general, different behavioral outputs may have different composition rules. In this work, we do not aim to contribute to component composition, as mechanisms for such composition have been described at length in prior work (e.g., see [60]). We mention it, however, to clarify the behavioral models that a twin synchronization protocol in a modern digital twin should have access to.

Operator intents: The operator specifies the behavioral system outputs of interest, together with tolerance for errors (in computing these behavioral outputs) attributed to weak synchronization. The specified behavioral outputs of interest determine the models to use for the selected components of the IoT system twinned. The error tolerance is used to decide how well various parameters need to be synchronized. More specifically, operator intents specify the vector of behavioral model outputs, Y, of interest to the application, and the corresponding vector of error bounds, ε that arise from imperfect synchronization. TwinSync must ensure that the twin's computed $\hat{Y}$ based on the values of parameters synchronized is maintained within ε from the Y that would have been computed if the twin had a perfect instantaneous view of all relevant model parameters.

3.3 DIGITAL TWIN HYPERVISOR

The digital twin hypervisor encompasses the aspects of the digital twin that are pertinent to the synchronization protocol. In order to support synchronization with multiple applications and configurations of the physical system, the digital twin needs to have a mechanism to understand the current IoT system configuration and customize its own modeling and synchronization accordingly. The components of that mechanism, as shown in Figure 3.1, are:

- *Query generator*: The query generator is responsible for (i) parsing operator intents, (ii) querying the system for its current configuration $G(V, E)$, (iii) selecting the pertinent behavioral component models from the library to model performance aspects

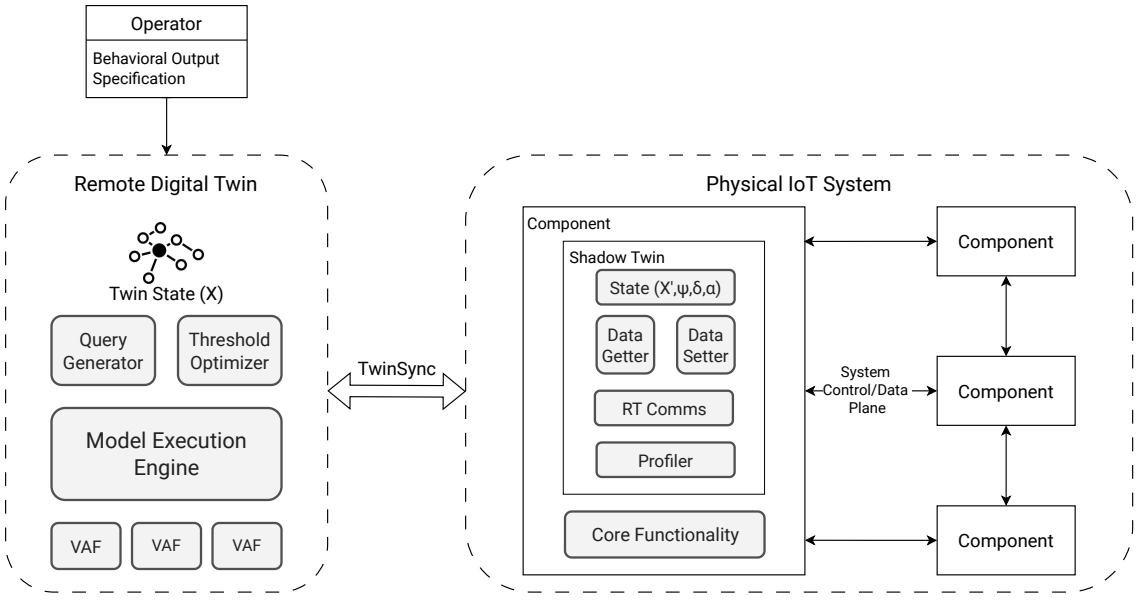

Figure 3.1: Digital Twin Architecture

relevant to the declared intents, and (iv) identifying the vector of all parameters, X, needed for these models. This is done as an initialization step every time the system is reconfigured or the operator specifies a new set of intents, Y. TwinSync will subsequently synchronize the vector X. We denote the copy of X on twin by $\hat{X}$. We shall also henceforth call the overall end-to-end model synthesized to relate Y and X by the function ϕ (i.e., $Y = \phi(X)$).

- *Synchronization policy optimizer*: For each element, x_i, of vector, X, the synchronization policy optimizer is responsible for calculating the conditions under which TwinSync should synchronize that element (i.e., send a copy of parameter x_i from the corresponding physical component in the IoT system to the twin). The design of this protocol is presented later in Chapter 4. These conditions are sent to the respective endpoint of TwinSync, called *shadow twin*, on each actual IoT device. The shadow twin thereafter checks for these conditions and sends a copy of the variable to the twin every time the conditions are satisfied.

- *Model Execution Engine*: The model execution engine updates its estimate of the behavioral output $\hat{Y}$ using the most recent values of variables synchronized, $\hat{X}$. While the twin does not have the most up-to-date values of X, TwinSync ensures that computing $\hat{Y}$ based on $\hat{X}$ produces a deviation that is bounded by ε between $\hat{Y}$ and the unknown ground truth value, Y.

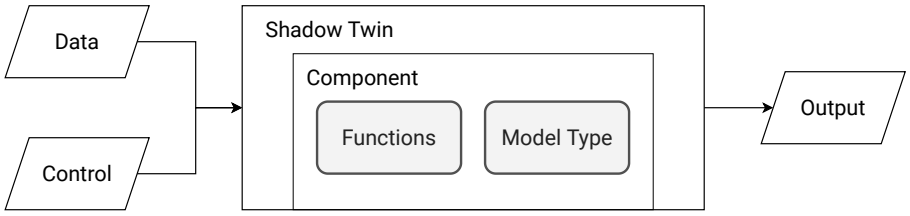

Figure 3.2: Shadow Twin Interface

- *Value-added Functions (VAF)*: Multiple value-added functions can be incorporated into the digital twin. Examples include anomaly detection, "what if" analysis, and system optimization. While these functions are tangential to the synchronization protocol, in this work, we demonstrate one such function, namely optimizing system configuration as will be shown in the evaluation.

3.3.1 Shadow Twin

The shadow twin is the counterpart of the digital twin that resides in the physical system. The twin and the physical system interact through the shadow twin. The shadow twin is implemented as a lightweight object wrapper that augments each system component that needs to be twinned, as shown in Figure 3.2. Each such component offers its independent core functionality and stores data pertaining to its instantiated models. The wrapper has access to the models and the real-time state of the component via an API that compliant components in the component library export. The component models can have multiple variants and the model corresponding to the operator-defined behavioral output is sent in response to the digital twin's initialization query. The shadow twin also is instantiated for every component in the physical system. The shadow twin's state consists of a subset of the physical system state X corresponding to that of the component it is associated with. It also stores the conditions used to decide if a variable should be synchronized (i.e., sent to the digital twin). It is responsible for copying the variable to the digital twin when these conditions are met. In addition, the shadow twin responds to the initialization queries generated by the digital twin's query generator. Importantly, the shadow twin includes a *profiler*. In the absence of pre-existing models for a component, the profiler may be used to generate such a model.

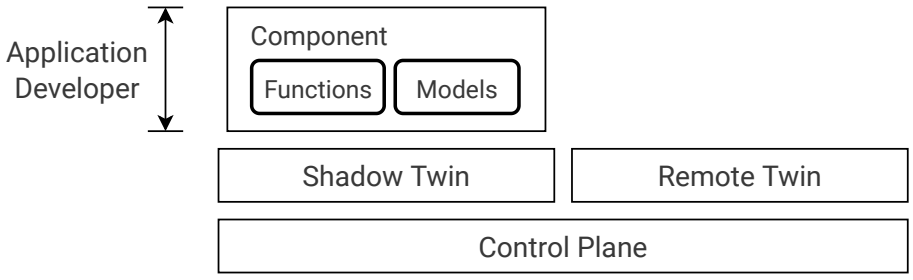

Figure 3.3: Layered Architecture

3.4 PHYSICAL SYSTEM - TWIN INTERFACE

The physical system is composed of modular components that can be connected to accomplish multiple tasks. A library of sensors, actuators, and other algorithms is predefined and available for configuration by the application developer. Additionally, it can also sample the input and output data and the control parameters. This is necessary for synchronization with the remote twin.

Communication between the remote twin and the shadow twin is separated into a data plane, managed by TwinSync, and a control plane that configuration queries and commands, as shown in Figure 3.3. The application developer only has control over the component and parameter definitions at the topmost layer of the stack. The data plane supports bidirectional communication between the remote twin and the shadow twin. The shadow twin can actuate configurations generated by the remote twin through the control plane.

CHAPTER 4: SYNCHRONIZATION PROTOCOL

In this chapter, we discuss the formulation of a synchronization protocol that can reduce the synchronization bandwidth overhead while honoring operator-defined accuracy constraints. We first formulate the weak synchronization problem as an optimization problem (Section 4.1), then derive a solution with optimality properties (under certain simplifying assumptions) (Section 4.2). The problem and solution are then generalized to the common case where similar variables are synchronized *in batch* (Section 4.3). We conclude with a description of the overall sequence of events in the synchronization workflow (Section 4.4).

4.1 WEAK SYNCHRONIZATION PROBLEM FORMULATION

Let us consider the state of the physical system at time, t, to be:

$$X_t = \{x_{c,i,t} | 1 \leq c \leq C, 1 \leq i \leq I\} \tag{4.1}$$

where $x_{c,i,t}$ denotes the value of the i^{th} parameter associated with the c^{th} component at time, t. Let $b_{c,i}$ denote the cost of synchronizing parameter $x_{c,i,t}$ (e.g., the size of the corresponding data structure in bytes, if communication is the bottleneck). At time, t, a behavioral output vector, Y_t, is the vector of quantities that the operator is interested in observing or optimizing, where (according to the behavioral models):

$$Y_t = \phi(X_t) \tag{4.2}$$

At the same time, the state of the system, as seen by the remote twin is denoted by:

$$\hat{X}_t = \{\hat{x}_{c,i,t} | 1 \leq c \leq C, 1 \leq i \leq I\} \tag{4.3}$$

where $\hat{x}_{c,i,t}$ denotes the value of the i^{th} parameter associated with the c^{th} component as seen by the digital twin at time t. It may differ from $x_{c,i,t}$ because synchronization is not immediate.

The behavioral output of interest can be estimated from $\hat{X}_t$ on the twin as follows:

$$\hat{Y}_t = \phi(\hat{X}_t) \tag{4.4}$$

Let b_i denote the cost of synchronizing $x_{c,i,t}$, meaning the cost of sending a copy of $x_{c,i,t}$ to the twin. We assume that the communication latency between the IoT devices and the

twin is substantially smaller than the rate at which state $x_{c,i,t}$ changes. Thus, ignoring communication latency, if parameter $x_{c,i,t}$ is sent to the twin at time, t_k, and then sent again later at time, t_{k+1}, we say that $\hat{x}_{c,i,t} = x_{c,i,t_k}$ for $t_k \leq t < t_{k+1}$. Let the total cost of synchronization (i.e., the sum of costs $b_{c,i}$ over all synchronization events) be denoted by *Cost*. We consider two flavors of the synchronization problem:

- *Minimum Cost, Bounded Error:* Develop a synchronization protocol that minimizes *Cost*, subject to the constraint that $|Y_t - \hat{Y}_t| \leq \varepsilon$, where ε is a specified tolerable error bound.
- *Minimum Error, Bounded Cost:* Develop a synchronization protocol that minimizes the error $|Y_t - \hat{Y}_t|$ subject to the constraint that $Cost \leq B$, where B is a specified bound derived, perhaps, from bottleneck resource capacity (e.g., available communication bandwidth).

4.2 BASIC SYNCHRONIZATION PROTOCOL

In our synchronization protocol, a variable $x_{c,i,t}$ is sent to the twin (i.e., is synchronized) when a certain synchronization condition is met. In the basic protocol, the condition is defined simply in terms of the deviation between the current value, $x_{c,i,t}$, and the previously synchronized value, $\hat{x}_{c,i,t}$. When that deviation reaches some error $\alpha_{c,i,t}$, the the variable is synchronized. Thus, the condition is expressed as:

$$|\hat{x}_{c,i,t} - x_{c,i,t}| \geq \alpha_{c,i,t} \tag{4.5}$$

To complete the description of the protocol, it remains merely to compute the optimal values of $\alpha_{c,i,t}$ that solve either the minimum cost, bounded error, or the minimum error, bounded cost problem, stated above.

To derive the basic synchronization protocol that solves either problem, we first observe, by computing the total derivative of Equation (4.2) with respect to X_t, that:

$$dY_t = \sum_{c,i} \frac{\partial \phi}{\partial x_{c,i,t}} * dx_{c,i,t} \tag{4.6}$$

or, approximately, for sufficiently small deviations, $\delta x_{c,i}$:

$$\delta Y_t = Y_t - \hat{Y}_t \approx \sum_{c,i} \frac{\partial \phi}{\partial x_{c,i,t}} * \delta x_{c,i,t} \tag{4.7}$$

Let $\gamma_{c,i}$ denote the (statistically) expected time interval during which variable $x_{c,i}$ changes by an amount, $\delta x_{c,i}$. The expectation of average synchronization cost for that variable is thus given by: $b_{c,i}/\gamma_{c,i}$. We can now derive a solution for the two optimization problems addressed in the preceding section, as described below.

4.2.1 Minimum Cost, Bounded Error

For every $x_{c,i}$ find the optimal $\delta x_{c,i}$ such that the following minimization problem is solved:

$$\min_{\delta x_{c,i}} \quad Cost = \sum_{c,i} \frac{b_{c,i}}{\gamma_{c,i}} \tag{4.8}$$

Subject to:

$$\delta Y_t = \sum_{c,i} \frac{\partial \phi}{\partial x_{c,i,t}} * \delta x_{c,i,t} \leq \varepsilon \tag{4.9}$$

4.2.2 Minimum Error, Bounded Cost

Alternatively, if the available bandwidth is not sufficient to maintain the above error bound, we can instead formulate a problem that minimizes error subject to the available bandwidth:

$$\min_{\delta x_{c,i}} \quad \delta Y_t = \sum_{c,i} \frac{\partial \phi}{\partial x_{c,i,t}} * \delta x_{c,i,t} \tag{4.10}$$

Subject to:

$$Cost = \sum_{c,i} \frac{b_{c,i}}{\gamma_{c,i}} \leq B \tag{4.11}$$

Here, B denotes the maximum bandwidth budget available for synchronization.

In either problem, once the optimal solution, $\delta x^*_{c,i}$, is found, the synchronization thresholds, $\alpha_{c,i}$, in Equation (4.5), are set to the optimal threshold found, or $\alpha_{c,i} = \delta x^*_{c,i}$.

4.3 GENERALIZATION TO BATCHED SYNCHRONIZATION

In practice, synchronizing individual variables, $x_{c,i,t}$, as suggested above, may be inefficient if the individual variables are short compared to such overheads as communication protocol packet headers and operating system buffer sizes. Thus, it is often advantageous to synchronize these variables in batches. The generalized version of the synchronization protocol extends the conditions under which a variable $x_{c,i,t}$ is sent to the twin to allow for batching. Specifically, let $X_{c,t} \in X_t$ denote the subset of all variables $x_{c,i,t} \in X_t$ that pertain to component c. In the generalized version, we assume the existence of some function $g_c(X_{c,t})$ that captures a higher-level state of the component, c, and can be computed on the IoT device side. All variables $x_{c,i,t} \in X_{c,t}$ (i.e., those pertaining to component c) are sent to the twin (i.e., synchronized) when the deviation δg_c between the current value of $g_c(X_{c,t})$ and the previously synchronized value, $g_c(\hat{X}_{c,t})$ reaches a threshold, say α_c^g. Thus, the generalized synchronization condition for the batch of variables $X_{c,t}$ is defined in terms of the component state as follows:

$$|g_c(\hat{X}_t) - g_c(X_t)| \geq \alpha_c^g \tag{4.12}$$

To derive the synchronization threshold, α_c^g, we must first express the relation between Y_t and X_t indirectly in terms of the state functions $g_c(X_{c,t})$ of individual components. Thus, we write:

$$Y_t = \phi^g(g_1(X_{1,t}), ..., g_C(X_{C,t})) \tag{4.13}$$

We then rewrite δY_t, in Equation (4.7), using the chain rule of derivatives, which leads to:

$$\begin{aligned} \delta Y_t &\approx \sum_c \frac{\partial \phi^g}{\partial g_c} \sum_i \frac{\partial g_c}{\partial x_{c,i,t}} * \delta x_{c,i,t} \\ &\approx \sum_c \frac{\partial \phi^g}{\partial g_c} \delta g_c \end{aligned} \tag{4.14}$$

which we can alternatively write as:

$$\delta Y_t \approx \sum_c M_c \delta g_c \tag{4.15}$$

where:

$$M_c = \frac{\partial \phi^g}{\partial g_c} \tag{4.16}$$

Note that, M_c above is simply the model of component c with respect to the behavioral output, Y_t. It describes how the behavioral output changes with changes in the value of the component state function, g_c. The above state decomposition by component is especially helpful if ϕ^g has a simpler derivative with respect to g_c than the derivative of the original function ϕ directly with respect to $X_{c,t}$. An example of such a scenario is given in the evaluation section. To find the optimal threshold, α_c^g (for the synchronization condition), the two problems described in the previous section can now be rewritten as follows.

4.3.1 Generalized Minimum Cost, Bounded Error

Since variables are now synchronized in batches, we define γ_c as the (statistically) expected time interval during which function $g_c(X_{c,i})$ changes by the threshold amount, δg_c. The generalized problem is thus to find, for every component, c, the optimal δg_c such that the following minimization problem is solved:

$$\min_{\delta g_c} \quad Cost = \sum_c \frac{\sum_i b_{c,i}}{\gamma_c} \tag{4.17}$$

Subject to:

$$\delta Y_t = \sum_c M_c \;\; \delta g_c \leq \varepsilon \tag{4.18}$$

4.3.2 Generalized Minimum Error, Bounded Cost

Alternatively, if the available bandwidth is not sufficient to maintain the above error bound, we can instead formulate a problem that minimizes error subject to the available bandwidth:

$$\min_{\delta g_c} \quad \delta Y_t = \sum_c M_c \;\; \delta g_c \tag{4.19}$$

Subject to:

$$Cost = \sum_c \frac{\sum_i b_{c,i}}{\gamma_c} \leq B \tag{4.20}$$

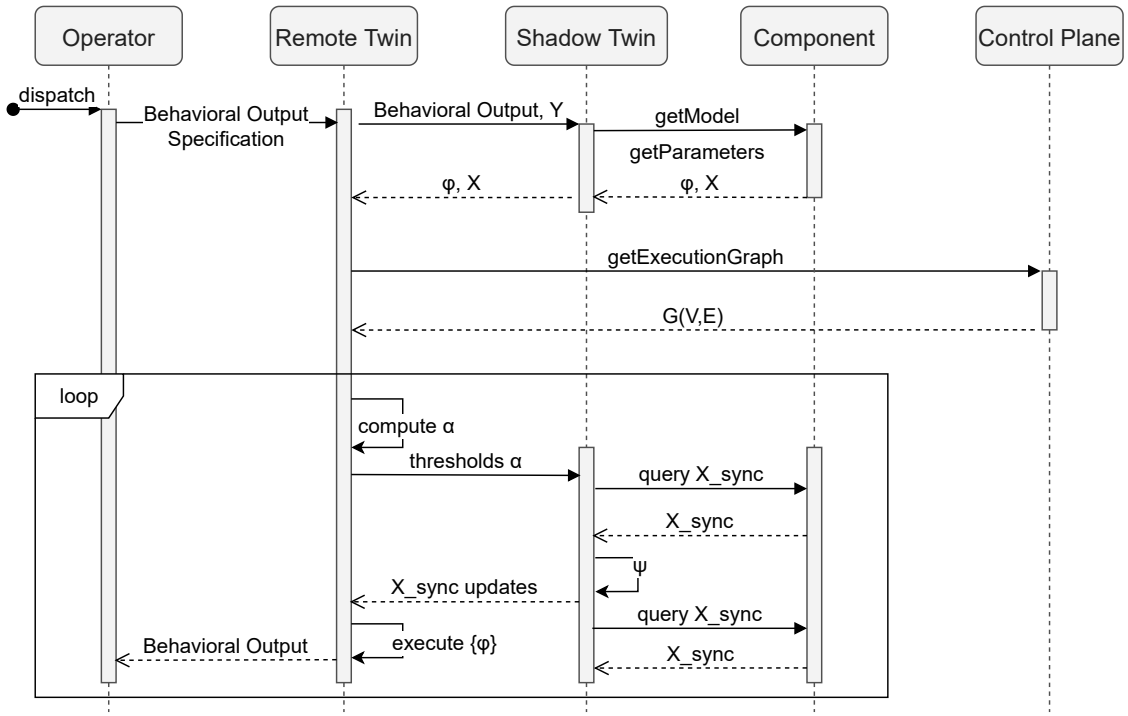

Figure 4.1: TwinSync Sequence Diagram

As before, B denotes the maximum bandwidth budget available for synchronization.

In either problem, once the optimal solution, δg_c^*, is found, the synchronization thresholds, α_c^g, in Equation (4.12), are set to the optimal threshold found, or $\alpha_c^g = \delta g_c^*$.

4.4 INTEGRATED WORKFLOW

Putting it together, the overall functionality of the synchronization framework is summarized below. We explain it in terms of the generalized protocol. The basic protocol is simply the special case where, $g_c = x_{c,i,t}$.

Given operator intents, Y, and error tolerances ε, the remote twin collects the behavioral models, M_c of deployed components and synthesizes function ϕ, such that $Y_t = \phi(X_t)$, or equivalently, its approximate linearization for small deviations, $\delta Y_t = \delta\phi(X_t)$, expressed in Equation (4.15). Optimal thresholds α_c^g are then computed for component state functions g_c. At every time t, the shadow twin at each node checks if any local physical system state g_c changed by more than the threshold, α_c^g, from its previously synchronized value. If so, then parameters $X_{c,t}$ are sent to the remote twin. The remote twin recomputes the real-time values of the estimated behavioral outputs, $\hat{Y}_t$, using Equation (4.4), or its approximate linearization, Equation (4.15).

Figure 4.1, depicts the communication between the various system entities during runtime. The operator and query protocols execute once during initialization whereas the others are in use throughout the system execution time. Algorithms 1 and 2 describe the operations executing on the remote twin and shadow twin in order to enable synchronization.

Algorithm 1 TwinSync (Shadow Twin)

```
1: queryComponentForModelsAndParameters(Y)
2: initializeThresholds(α_i)
3: while true do
4:     for each x_i in X_sync do
5:         syncDecision ← synchronizationDecision()
6:         if syncDecision is True then
7:             synchronizeRemote(x_{c,i,t})
8:         end if
9:         if receivedUpdate is True then
10:            reconfigureThreshold({α_i})
11:        end if
12:    end for
13: end while
```

Algorithm 2 TwinSync (Remote Twin)

```
1: queryShadowTwinState()
2: queryExecutionGraph()
3: initializeTwinState()
4: computeAndSendThreshold({α_i})
5: while true do
6:     behavioralOutput ← computeBehavioralOutput()
7:     if receivedUpdate is True then
8:         computeAndSendThreshold({α_i})
9:     end if
10: end while
```

CHAPTER 5: EVALUATION

In this chapter, we analyze the feasibility of synchronizing twin state with practically low bandwidth requirements while simultaneously satisfying required accuracy constraints. We perform a two-fold evaluation to analyze the performance of TwinSync along multiple axes. Our first approach involves utilizing synthetic functions and data to verify the protocol in diverse setups (Section 5.1). We then conduct a case study on a real system to analyze the performance gains obtained by utilizing the twin to support application-specific system optimization (Section 5.2). We conclude with a discussion on various aspects of the system implementation and evaluation (Section 5.3).

5.1 SYNTHETIC EVALUATION EXPERIMENTS

For the synthetic evaluation, we use hypothetical functions, ϕ, to evaluate the accuracy and bandwidth trade-off of TwinSync relative to a baseline. Table 5.1 gives an overview of the experiments performed.

Table 5.1: Synthetic Evaluation

ID	Experiment Type	Parameter Count	ε	r	b	B_{max}	Buffer Size
1	Linear	3	5	$0.4, 0.2, 0.1$	$2, 2, 2$	2	4
2	Linear	3	7	$0.4, 0.2, 0.1$	$2, 2, 2$	2	4
3	Linear	3	9	$0.4, 0.2, 0.1$	$2, 2, 2$	2	4
4	Non Linear	3	1	$0.004, 0.002$	$2, 2$	2	4
5	Non Linear	3	3	$0.004, 0.002$	$2, 2$	2	4
6	Non Linear	3	5	$0.004, 0.002$	$2, 2$	2	4
7	Logarithmic	3	0.2	$40, 20, 10$	$2, 2, 2$	4	4
8	Logarithmic	3	2	$40, 20, 10$	$2, 2, 2$	4	4
9	Logarithmic	3	5	$40, 20, 10$	$2, 2, 2$	4	4
10	Linear (Trade off)	3	2	$0.4, 0.2, 0.1$	$2, 2, 2$	1, 2, 4, 6	4
11	Non Linear (Trade off)	2	2	$0.004, 0.002$	$2, 2$	1, 2, 4, 6	4

5.1.1 Experimental Set-up

The experimental setup includes multiple nodes which are taken as the components present in the TwinSync architecture. For every experiment, the *behavioral model* (of the IoT system) is an equation that belongs to a particular class (called Experiment Type in Table 5.1), with multiple parameters (denoted by column 3 in Table 5.1). The input to the model are

Table 5.2: Average Error Per Experiment Type

Experiment Type	Avg TwinSync Error	Avg Baseline Error
Linear	1.45767	11.29499
Non Linear	0.90455	1.90963
Logarithmic	0.23085	0.80833

variables generated by a random walk with the step size indicated by r in Table 5.1. Each parameter is also associated with a bandwidth cost denoted by b which indicates the synchronization penalty. Every experiment also has a maximum bandwidth budget B_{max} that constrains the number of parameters that can be synchronized with the twin within a single time step. Overloading the budget is thus penalized by postponing the excess data to a later time step, as such data gets queued for re-transmission.

5.1.2 TwinSync-based Digital Twin

The digital twin is provided with an error bound denoted by ε in Table 5.1. Its objective is to synchronize state such that this error bound is not exceeded. In order to accomplish this, the twin uses TwinSync's threshold-based synchronization protocol to inform the system of synchronization thresholds for each of the parameters. We run two versions of the synchronization algorithm: the minimum cost, bounded error version, and the minimum error, bounded cost version, and choose the output that satisfies the constraints.

5.1.3 Baseline

Most digital twinning solutions rely on simple protocols that do not take into consideration bandwidth cost or application semantics and transfer the entirety of the state. In order to contrast the accuracy and bandwidth consumption of using TwinSync with such protocols, we set up an additional baseline to measure the accuracy and bandwidth observed when all parameters are synchronized at every time step.

The set of experiments and their various characteristics are shown in Table 5.1. Figure 5.1 has plots depicting the outcome of all the experiment types. Figure 5.1(a) and (d) show results for linear experiments, Figure 5.1(b) and (e) show results for nonlinear experiments, and Figure 5.1(c) and (f) show results for logarithmic experiments. Both categories of plots are compared with the baseline when run under identical conditions.

We observe that with the same settings, the TwinSync-based twins are able to generate

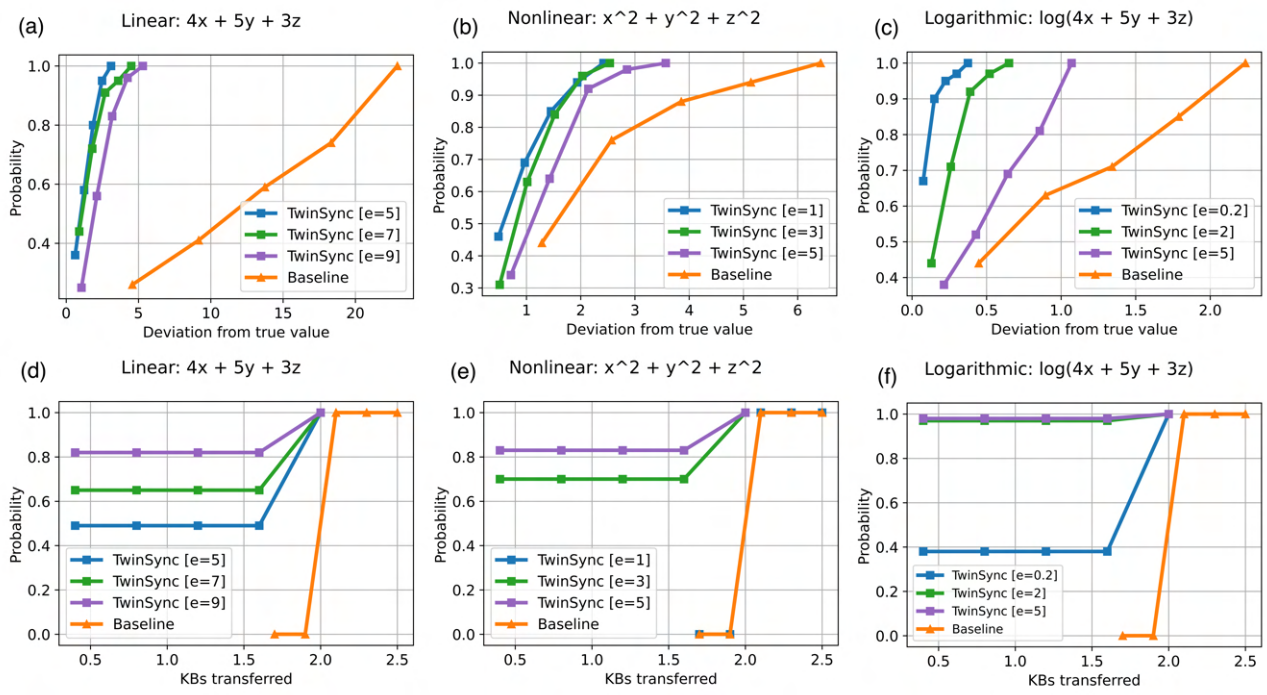

Figure 5.1: Synthetic Evaluation Experiments: (a) Cumulative Error for Linear Experiments, (b) Cumulative Error for Non-linear Experiments, (c) Cumulative Error for Logarithmic Experiments, (d) Cumulative Bandwidth for Linear Experiments, (e) Cumulative Bandwidth for Non-linear Experiments, (f) Cumulative Bandwidth for Logarithmic Experiments

behavioral output values that have a magnitude of error less than that generated by the baseline. As the error bound is increased the TwinSync-based twins are able to reduce the bandwidth consumption in addition to having a lower deviation from the true value than the baseline. The average error values for each experiment type are shown in Table 5.2 and we observe that the error captured while using TwinSync is significantly lower than the baseline across all experiment types.

We conduct experiments 10 and 11 to observe the variation in error as we increase the bandwidth budget, B_{max}. The average error is lower while using TwinSync and later becomes a stable constant for both the baseline and TwinSync. The minimum error for TwinSync is higher but it remains within the specified error bound ε and uses lesser bandwidth than the baseline to maintain that state. This analysis demonstrates that TwinSync is able to represent the physical system state with significantly lower bandwidth overhead while satisfying an error bound.

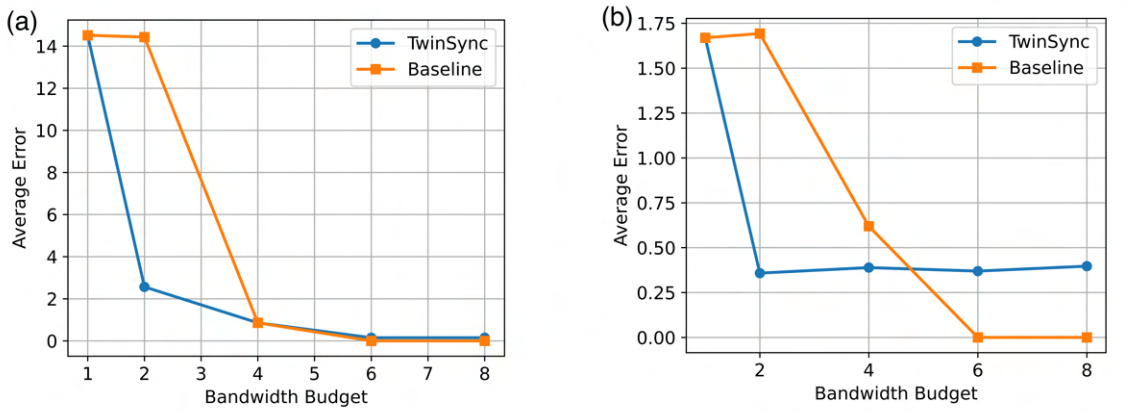

Figure 5.2: Accuracy-Bandwidth Trade-off: (a) Linear Experiments, (b) Non-linear Experiments

5.2 A CASE STUDY

To illustrate the benefits of TwinSync we describe how it can integrate with and twin a simple application where acoustic sensors, seismic sensors, and cameras are deployed to detect and classify types of passing vehicles.

5.2.1 System Implementation and Execution Loop

We used the system described in IoBT OS [13]. This system was developed for battlefield IoT applications [61], where sensors with limited machine intelligence are deployed in the field and tiered into two categories: (i) simple sensors that act as *triggers* when target presence is suspected, and more accurate (ii) confirmation sensors that wake up when triggered to investigate the current environment but need more resources. Each sensing node was a Raspberry Pi [62] equipped with two simple (trigger) sensors: namely, a vertical-axis geophone and a microphone. Their measurements were processed locally by a simple classifier developed in earlier work for intelligent IoT applications [63]. Each node was also equipped with a camera that could be activated on demand to take pictures of the scene and send these pictures to a more powerful edge server for processing and visual confirmation using a version of YOLO (a vision-based object detector) [64]. The end-to-end pipeline used in this case study is depicted in Figure 5.3.

The system is implemented using Robot Operating System 2 (ROS2) [65] and each function (classifier, image detection, sensor handlers) is modeled as an independent node. The remote twin is implemented as a standalone node that is launched on the remote server. The shadow twin is a library that can be imported into nodes that are required to be twinned and

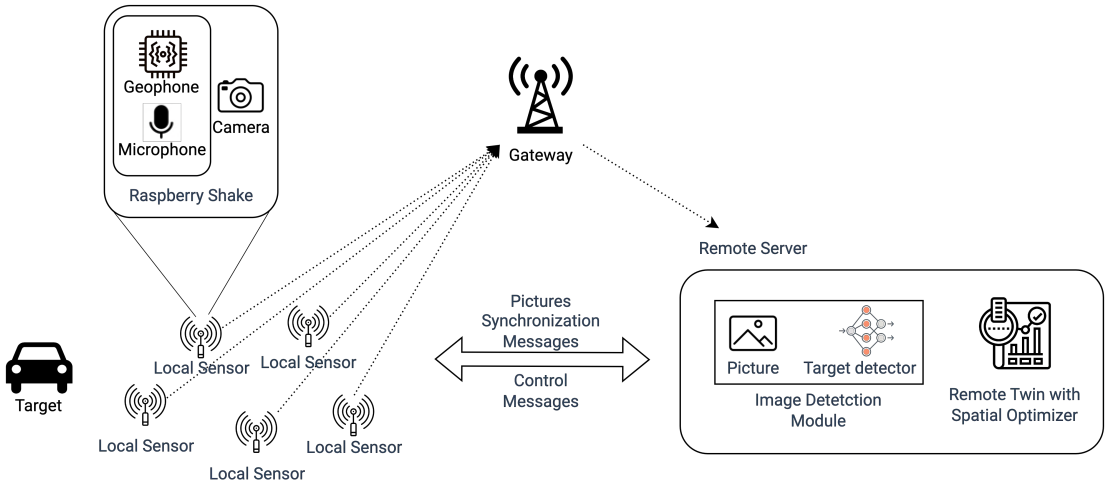

Figure 5.3: Experimental Pipeline for the Case Study

require no additional customization in the application layer of the system. Our implementation makes it very easy to add or remove digital twin functionality at the node level. The remote twin also supports a membership and heartbeat service to keep track of node status. In order to reduce the synchronization overhead all shadow twin updates are published on a single topic (communication channel) but remote twin updates are published on individual remote twin-shadow twin topics which also helps avoid reception by unintended nodes.

5.2.2 Digital Twin Formulation

To illustrate how the synchronization protocol may be used, let us consider a simplified view where the purpose of synchronization is simply to "twin" data from trigger sensors sufficiently for the twin to reproduce the target detections seen by the original system. Given the capability to reproduce such detections on the twin, it will be possible to use the twin for different purposes such as updating system configuration to suppress redundancy or detecting anomalous phenomena related to target detection. With that in mind, we consider the behavioral output Y_t to be the probability of a target being present, given the trigger measurement vector, X_t. Thus:

$$Y_t = \phi(X_t) = P(\texttt{Target}|X_t) \tag{5.1}$$

The above is a simple (illustrative) instantiation of Equation (4.2) in the context of our application. For the purposes of this example, let the system be composed of C nodes

(each viewed as a component, c) and let each node periodically produce data batches, where the data batch produced at time, t, is called $X_{c,t}$. The batch is composed of the latest I successive sensor samples from node c. For example, a seismic sensor with a sampling rate of $100Hz$ might generate a data batch of 200 samples every two seconds. Consistently with the notations used earlier in Chapter 4, at time t, we define X_t as the set of all variables produced at time t. Further, we denote by $\hat{X}_t$ the latest version of X_t synchronized with the twin by time, t. $\hat{Y}_t$ is the estimate of Y_t computed at the twin from $\hat{X}_t$. Substituting in Equation (4.7) from Equation (5.1), the error $\delta Y_t = Y_t - \hat{Y}_t$ is given by:

$$\delta Y_t \approx \sum_{c,i} \frac{\partial P(\texttt{Target}|X_t)}{\partial x_{c,i,t}} * \delta x_{c,i,t} \tag{5.2}$$

In practice, computing the above partial derivatives precisely is as hard as performing the actual target detection, so some approximation is unavoidable. We take the simplified view that the probability of target presence at a node, c, and at a time, t, is some nonlinear function $\phi^g(E_{c,t})$ of sensor signal energy, $E_{c,t}$. Thus, using the chain rule of derivatives, Equation (5.2) can be rewritten as:

$$\delta Y_t \approx \sum_{c} \frac{d\phi^g(E_{c,t})}{dE_{c,t}} \sum_{i} \frac{\partial E_{c,t}}{\partial x_{c,i,t}} * \delta x_{c,i,t} \tag{5.3}$$

where energy, $E_{c,t}$ is defined as:

$$E_{c,t} = \sum_{i} x_{c,i,t}^2 \tag{5.4}$$

Substituting from Equation (5.4) into Equation (5.3), we get:

$$\begin{aligned} \delta Y_t &\approx \sum_{c} \frac{d\phi^g(E_{c,t})}{dE_{c,t}} \sum_{i} (2x_{c,i,t}) \delta x_{c,i,t} \\ &\approx \sum_{c} \frac{d\phi^g(E_{c,t})}{dE_{c,t}} \ \delta E_{c,t} \end{aligned} \tag{5.5}$$

where $\delta E_{c,t}$ is simply the change in signal energy at node c and time t (from the previous window). This is, in fact, the generalized Equation (4.14) with $g_c \equiv E_c$. In other words, each component, c, will thus have a model:

$$M_c = \frac{d\phi^g(E_{c,t})}{dE_{c,t}} \tag{5.6}$$

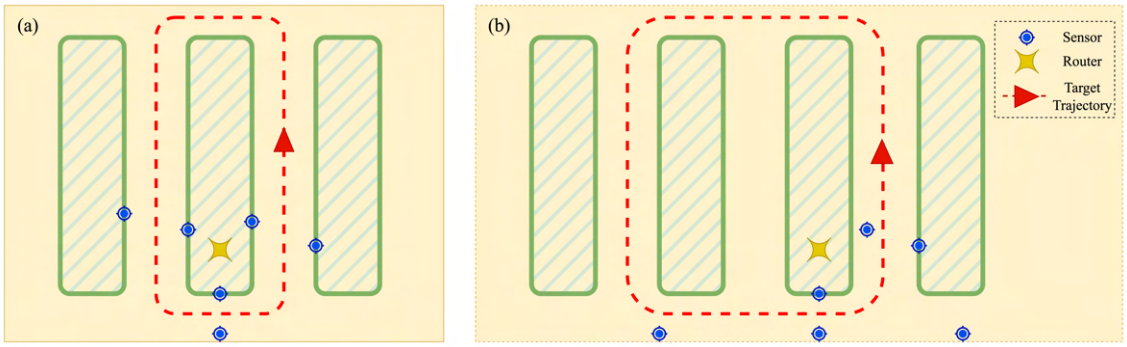

Figure 5.4: Experimental Setup for Data Collection: (a) Setup for Training Dataset Collection. (b) Setup for Evaluation Dataset Collection

Using the generalized formulation described in Section 4.3.1 or Section 4.3.2, we can now derive the optimal values of δE_c (for each node, c), denoted α_c^g, for use as a synchronization condition (per Equation (4.12)), to decide when to synchronize each data batch, $X_{c,t}$. Specifically, to solve the generalized minimum cost, bounded error problem we use Equations (4.17) and (4.18), whereas to solve the generalized minimum error, bounded cost problem we use Equations (4.19) and (4.20).

5.2.3 Data Collection and Profiling

In order to perform the computations described above we collect a dataset that can be profiled to construct the model $M_c = d\phi^g/dE_c$. The experimental topology details are shown in Figure 5.4. We collect two versions of data, the training set is used for training the classifier as well as profiling whereas the evaluation data was exclusively used for playback and evaluation. In our experiment, six multimodal sensing nodes were used and arranged in different positions during training and evaluation data collection. The sensor locations were constrained by the range of the router. We drove three different cars near the sensors to serve as the target. Table 5.3 provides a description of the dataset.

Table 5.3: Dataset Description

Vehicle	Train Set	Evaluation Set
2022 Mazda MX5	8.07 miles, 30m 45s	3.39 miles, 13m 54s
2016 Nissan Rogue	7.85 miles, 30m 44s	3.30 miles, 13m 58s
Ford Mustang	7.92 miles, 30m 23s	4.08 miles, 14m 9s

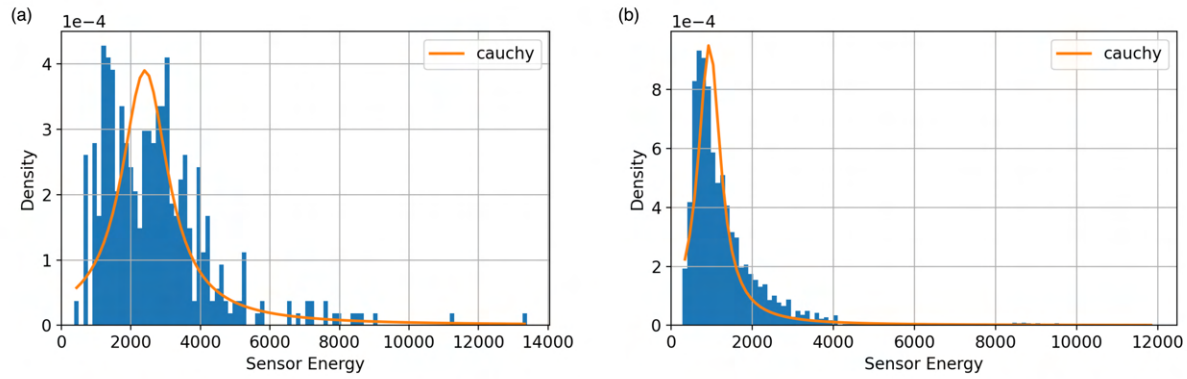

Figure 5.5: Distribution of Seismic Energy: (a) When the Target is In-Range (b) When the Target is Out-of-Range

In order to profile ϕ^g we process the training data to derive distributions of the sensor energies when the target is in range (within *50 feet* of the sensor) out of range. These distributions are used to compute the probability of target presence given energy derived from the law of total probability:

$$\phi^g = \frac{P(E_c^s|\texttt{Target}).P(\texttt{Target})}{P(E_c^s|\texttt{Target}).P(\texttt{Target}) + P(E_c^s|\neg\texttt{Target}).P(\neg\texttt{Target})} \tag{5.7}$$

In order to compute the individual probabilities in Equation (5.7) analyze the distributions to fit them to a Cauchy distribution since it has the lowest sum of squared errors as shown in Figure 5.5. This equation serves as the behavioral output function ϕ^g. The empirically found ϕ^g at an energy E_c^s is derived from the parameters of the distribution and given by the equation below:

$$\phi^g = \frac{f(1689.8, 583.3, X_c^s) * 0.4}{f(955.8, 283.9, X_c^s) * 0.6 + f(1689.8, 583.3, X_c^s) * 0.4} \tag{5.8}$$

where f denotes the density of the Cauchy distribution,

$$f(l, s, x) = \frac{1}{\pi} \arctan\left(\frac{x-l}{s}\right) + \frac{1}{2} \tag{5.9}$$

where X_c^s refers to the data batches from the seismic sensor, respectively, at node c. Additionally, the rate of change of the signal energy is found to be empirically approximated

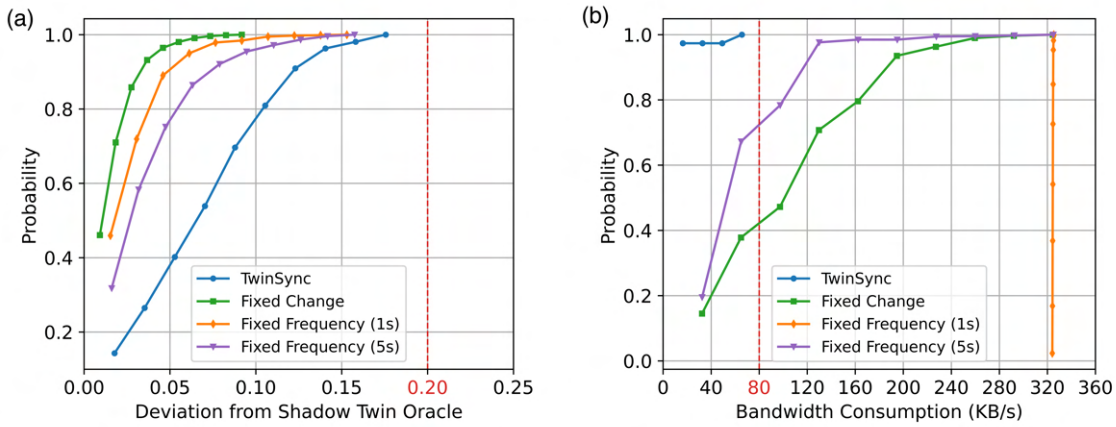

Figure 5.6: Bounded Synchronization for Computing Target Detection: (a) Cumulative Error Distributions when the Error Bound is 0.2, (b) Cumulative Bandwidth Distributions when the Bandwidth Budget is 80KB/s

by:

$$\gamma_c^s = 0.0157(E_c^s/1000)^4 - 0.857(E_c^s/1000)^3 + 8.97(E_c^s/1000)^2 + 0.339(E_c^s/1000) + 327 \tag{5.10}$$

Evaluating it with the corresponding synchronization threshold α_i gives us the expected latency γ. This completes the preparation of the values needed to solve the optimization problem.

5.2.4 Twinning Experiment and Baselines

As suggested above, the twin computed energy periodically, if the energy change exceeded the threshold pre-computed by the optimization algorithm, sensor buffer values were shared with the twin. We contrast our synchronization protocol with a *fixed frequency* twin synchronization approach, where every parameter has to be sent to the twin at a set interval. We also deploy a *fixed change* mode of synchronization where the synchronization thresholds are the same fixed values for all parameters. In our experiments, we use a deviation of 200 based on the statistical distribution of the trace data. These two synchronization modes are incorporated to demonstrate the pitfalls of using a simplified version of the TwinSync synchronization protocol. The error is calculated with respect to an ϕ^g oracle value which is obtained by computing and aggregating the outputs from the real system (shadow twin).

Deeper analysis with digital twin-specific customizations of the wireless sensor network synchronization approaches in Chapter 2 is left to future work.

Figure 5.6(a) shows the cumulative distribution of the error for both TwinSync and the other synchronization algorithms 5.2.4 when the error bound is set to 0.2. We observe that all approaches have errors well within the bound. TwinSync has a slightly higher average error but is able to accomplish this by using significantly lesser bandwidth. Figure 5.6(b) shows the distributions when the error bound is set to 0.25 but the bandwidth budget is constrained to $80KB/s$. We observe that TwinSync is able to bound the bandwidth consumption with a looser error bound whereas the baselines fail to limit the bandwidth consumption. Table 5.4 compares the target detection capabilities of DeepSense and the ϕ^g function running on the shadow twin (oracle) based on the detections made per energy bin. We observe that ϕ^g is more sensitive to higher sensor energies which may be due to the profiling mechanism where higher energy tends to be associated with the target being near.

Table 5.4: Comparison between DeepSense and ϕ^g

Energy Bins	≤ 1828	≤ 3371	≤ 4913
DeepSense Detections	1343	186	47
ϕ^g Detections	1114	753	198
Percentage Difference	0.17	0.75	0.76

5.2.5 Value Added Twin Functions

The above section described how to twin-component state in order for some estimated behavioral output on the twin to remain within an error bound from its real value in the system. In this section, we give an example of other value-added functions that can now be implemented on top. Specifically, we consider configuration optimization.

In our application, since trigger sensors on multiple sensing nodes may detect and act on the same target, redundant images may be sent to the edge server with requests for visual confirmation. To prevent such redundant images from draining network bandwidth, redundant nodes need to coordinate properly to send only one request to the server per redundant group. This is a configuration optimization problem that we can delegate to the digital twin. In such an architecture, the twin may analyze the correlation in acoustic and seismic signatures among different nodes to decide which nodes are largely redundant. The answer may depend on the specific deployment pattern and typical target trajectories and thus may differ from deployment to deployment. Depending on the discovered correlations in the current deployment, the system can be configured such that (all but one) redundant

Figure 5.7: Target Detections made by three different sensors

confirmation requests are "muted". Only one of the redundant nodes is configured to send images for visual confirmation, whereas the transmission of images from other nodes is suppressed.

Note that, the redundancy patterns can change as target mobility profiles change. Thus, the sets of redundant nodes need to be continually reassessed. The proposed TwinSync protocol already replicates sensor readings, $X_{c,i}$. On the twin, a value-added function can thus optimize system configuration accordingly. The twin can update the system configuration via the control plane.

The above system setup is shown in Figure 5.3. Note that, in this system, no action needs to be taken unless a target is nearby. Thus, we need to model target detection probability, which we take as the behavioral output, Y.

As observed, the twin is able to synchronize the necessary state and compute the metric with reasonable accuracy and minimal overhead. In Figure 5.8 we study the system performance by simulating different bandwidth capacities. We analyze the spatial optimizer module's effectiveness in reducing redundant transfers. This can be measured by the image loss rate which quantifies the loss in images sent after a confident DeepSense detection to the remote server for further processing. If the image loss rate is high the overall performance of the image detection module is impacted as it does not receive pictures in a timely manner for decision-making. We observe that the reduction in redundant transfers results in a decrease of up to 10% in image loss at lower bandwidth budgets over a baseline that does not use any form of optimization. Figure 5.7 shows a sample of the images received by YOLO running on the remote server for classification.

Our case study demonstrates TwinSync can be customized to support several such value-added functions that can enhance system performance with minimal overheads in addition to mirroring the system state.

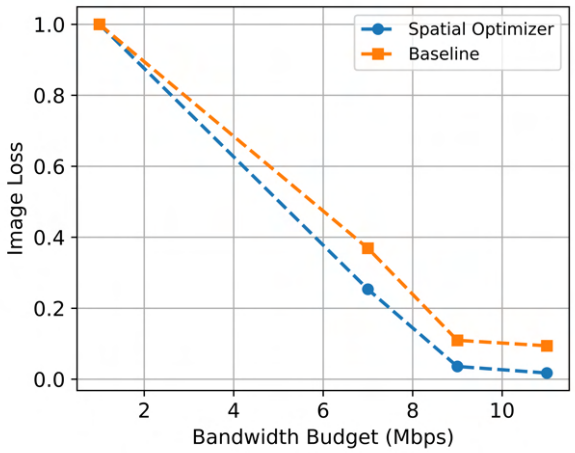

Figure 5.8: Impact of the Spatial Optimizer on the Image Loss

5.3 DISCUSSION

The evaluation provides us with a holistic understanding of the performance and capabilities of TwinSync. However, there are certain challenges that arise due to practical and hardware limitations which we discuss below.

5.3.1 Issues with Data Misalignment

The IoBT system was implemented and designed in such a way that it can support three modes of operation:
Record: In this mode, the seismic, acoustic, and image data is collected at a fixed frequency and stored on disk. The sensor buffer size at every timestep depends on the sampling rate.
Playback: In this mode, the recorded data is replayed by the sensor nodes. The playback rate is the same as the recording rate. Downstream recorded data such as camera images are published by finding the entry that was captured closest to the offset relative to the system start time.
Real-time: This is the live operation mode of the system where data is published and processed in real time.

The system supports all three modes of operation however, for the case study the record and playback mode was used for data consistency across TwinSync and the baselines. ROS2 supports the bagging and playback of data however we add custom support for these operations due to scale issues. This mode of experimentation gave rise to some interesting challenges due to clock synchronization issues. Firstly, the Raspberry Pi's do not have access to the internet when deployed in the experimental setting. As a result, they are unable

to synchronize their clocks. To mitigate this issue the time on the boards is explicitly set to the server time at the beginning of the experiment. During profiling and playback, we observed that there was a varying amount of offset (in the order of seconds) among the sensor nodes, server, and the GPS data collected for the target. In order to overcome this discrepancy we had to cross-correlate the target distance signal and the sensor reading to compute the lag between the signals and align the data. The consequence of having misaligned data is that system and the twin might detect targets after it has passed invalidating the results. Another option that we explored was the use of GPS modules on every sensor board but this solution disrupts the microphone data due to a shared USB buffering mechanism in Raspberry Pi.

5.3.2 Hardware Issues

Each node comprises multiple sensors. During the field deployment, we observed various hardware issues impacting the quality of data collected by the system. The seismic sensor readings are impacted by the power supply to the Raspberry Shake. The Raspberry Pi board, even when in a power-on state can experience voltage fluctuations resulting in inaccurate readings. These fluctuations can be caused due to loose connections or issues with the cable and are hard to manually detect. One mitigation was to manually measure the voltage on every node before starting the experiment. We also observe that the microphone readings are very noisy due to the high sensitivity of the device. The use of alternate microphones might be one potential solution to this issue.

5.3.3 Issues with the Data Distribution Service

The bandwidth bottleneck in the system is largely due to the images being sent from the nodes to the remote server for processing. The default data distribution service (DDS) used by ROS2 was not able to handle the size or frequency of image transfers generated by the system. We utilize socket layer communication for image transfers in order to support more efficient communications. The bandwidth strain highlights the benefits of using tools such as the spatial optimizer supported on the twin to alleviate resource contention in restrictive IoT frameworks.

5.3.4 Data Freshness Requirements on the Twin

The freshness of the data synchronized by the twin is driven by the bandwidth budget and error bound specified. Value-added functions with real-time data requirements might require tighter error bounds or end up operating on stale data producing inaccurate results. In such cases, the TwinSync protocol will likely be reduced to the trivial baseline where data has to be synchronized very frequently. However, TwinSync is performant in low-bandwidth conditions and can mitigate staleness caused due to network congestion.

CHAPTER 6: CONCLUSION AND FUTURE WORK

In this chapter we present future research directions that can be pursued as direct extensions to the work discussed in this thesis (Section 6.1). We then conclude by highlighting the salient contributions of this work (Section 6.2).

6.1 FUTURE WORK

6.1.1 Improvements to the Spatial Optimizer

One dimension of the evaluation that would benefit from further analysis is the accuracy trade-off associated with the use of the spatial optimizer. A reduction in the images sent by the nodes in the server can result in missing the target completely if the wrong node is suppressed. However, the measurement of accuracy is not very straightforward due to the indirection introduced by the twin. For instance, there may be cases where the system does not detect the target even in the absence of the spatial optimizer, or there may be cases where only the spatial optimizer results in an image with a successful detection. The accuracy is thus dependent on many factors such as camera orientation, end-to-end latency, and so on. We hope to explore this trade-off along multiple dimensions in future work.

6.1.2 Library of Value-Added Functions

Value-added functions are modules that run on the remote twin and operate on the twin state to generate various insights and actions. There can be numerous value-added functions supported on the digital twin simultaneously. In this work, we study a configuration-based function through the spatial optimizer. The modules could also be passive components such as a data store where sensor readings corresponding to certain events are offloaded to the twin for future analysis. This is especially useful in IoT devices with limited computing and memory resources. The twin could also support anomaly detection where it can learn a baseline metric and observe node-level inconsistencies. This setup would be helpful in the presence of an adversary but can also help with monitoring the health of the sensors which we found to be a major problem in our experiments. An important consideration is the construction of appropriate behavioral output specifications that can encompass the data requirements of all the modules.

6.1.3 Privacy and Security Considerations

In this work we focus on optimizing communication to save bandwidth costs. However, the cost in our formulation could be associated with numerous other metrics like power, information sensitivity, and so on. IoT systems are often used in healthcare and battlefield scenarios where the protection of data is critical due to its sensitive nature. Data privacy is especially important in such cases to be compliant with various laws and avoid leakage to adversaries. It would be helpful to explore a formulation where the granularity of the data being synchronized is taken into consideration and analyze its trade-offs with accuracy.

6.1.4 Multiple Behavioral Outputs

Behavioral outputs capture the operator's intent and provide context to the synchronization protocol. They are represented by a metric or view of the system that is of importance to the operator and requires twinning within an error bound. Our experiments use a single behavioral output specification i.e., target detection. TwinSync can support multiple behavioral outputs. We could explore the alternate optimizations that can be leveraged when the twin is synchronizing for multiple objectives. A straightforward approach would be to independently compute thresholds and take the minimum across all behavioral outputs. An interesting direction would be to study a multi-objective optimization scenario where further complexities arise while choosing a trade-off point in the Pareto front.

6.2 CONCLUSION

In this thesis, we studied the question of whether it is possible to synchronize twin state with practically low bandwidth requirements while simultaneously satisfying required accuracy constraints. To evaluate this question, we propose a synchronization framework for IoT digital twins that can overcome some of the communication challenges IoT deployments face with regard to bandwidth constraints. We present a TwinSync, a configurable and extensible system architecture that supports our synchronization mechanism and is robust to application-level modifications which are common in IoT systems. Our synchronization algorithm takes the operator's intent for digital twinning into consideration and uses sensitivity analysis to generate a synchronization schedule within the system's bandwidth constraints.

In Chapter 2 we discuss recent work in the digital twinning and approximate synchronization space. We also discuss the complexities of building digital twins for IoT systems and highlight the need for an approach that is sensitive to the operator's requirements while

being considerate of the limited resource availability. In Chapter 3 we present an architecture for constructing and synchronizing digital twins in a flexible manner that is robust to reconfigurations common in IoT systems. We present a set of simple abstractions that can be defined and customized independent of the underlying IoT application. In Chapter 4 we present the formulation of the synchronization protocol. We initially define a basic version of the protocol and then extrapolate that to a generalized batched synchronization that can make better utilization of bandwidth.

Through our evaluation in Chapter 5, we observe that TwinSync is indeed able to synchronize the state needed for digital twinning with practically low bandwidth overhead. The evaluation is performed using both synthetic data and by integrating it with a case study application. The synthetic evaluation provides an analysis of TwinSync subject to diverse analytical models and bandwidth budgets. We observe that TwinSync can bound the error and the bandwidth overhead according to changing network conditions and application requirements. As the error bound is loosened the bandwidth overhead decreases. The results from the case study show that using TwinSync supports the synchronization of digital twins at a bounded accuracy while utilizing lesser than 50% of the bandwidth overhead of other methods. The spatial optimizer is able to utilize the state synchronized by the twin to reduce the image loss due to network congestion by 10%. Our case study demonstrates how TwinSync can support a real-world IoT system to accomplish its mission more effectively through intelligent optimal re-configurations.

In conclusion, this thesis provides a description of a framework that serves as a promising step towards building and synchronizing digital twins that can address the resource constraints characteristic of IoT systems. We present a synchronization protocol that optimizes the bandwidth overhead required for digital twin synchronization. We hope that this methodology can be expanded to develop robust and resilient digital twins for IoT systems that can operate in limited-resource environments.

REFERENCES

[1] S. Bagchi, T. F. Abdelzaher, R. Govindan, P. Shenoy, A. Atrey, P. Ghosh, and R. Xu, "New Frontiers in IoT: Networking, Systems, Reliability, and Security Challenges," *IEEE Internet of Things Journal*, vol. 7, no. 12, pp. 11 330–11 346, 2020.

[2] A. Albahri, J. K. Alwan, Z. K. Taha, S. F. Ismail, R. A. Hamid, A. Zaidan, O. Albahri, B. Zaidan, A. Alamoodi, and M. Alsalem, "IoT-based Telemedicine for Disease Prevention and Health Promotion: State-of-the-Art," *Journal of Network and Computer Applications*, vol. 173, p. 102873, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804520303374

[3] E. Baccour, N. Mhaisen, A. A. Abdellatif, A. Erbad, A. Mohamed, M. Hamdi, and M. Guizani, "Pervasive AI for IoT Applications: A Survey on Resource-Efficient Distributed Artificial Intelligence," *IEEE Communications Surveys Tutorials*, vol. 24, no. 4, pp. 2366–2418, 2022.

[4] G. Alexakis, S. Panagiotakis, A. Fragkakis, E. Markakis, and K. Vassilakis, "Control of Smart Home Operations Using Natural Language Processing, Voice Recognition and IoT Technologies in a Multi-Tier Architecture," *Designs*, vol. 3, no. 3, 2019. [Online]. Available: https://www.mdpi.com/2411-9660/3/3/32

[5] F. Shrouf, J. Ordieres, and G. Miragliotta, "Smart Factories in Industry 4.0: A Review of the Concept and of Energy Management Approached in Production Based on the Internet of Things paradigm," in *2014 IEEE International Conference on Industrial Engineering and Engineering Management*, 2014, pp. 697–701.

[6] E. Manavalan and K. Jayakrishna, "A Review of Internet of Things (IoT) Embedded Sustainable Supply Chain for Industry 4.0 Requirements," *Computers Industrial Engineering*, vol. 127, pp. 925–953, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835218305709

[7] 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*, vol. 108, no. 10, pp. 1785–1824, 2020.

[8] Z. Jiang, Y. Guo, and Z. Wang, "Digital Twin to Improve the Virtual-Real Integration of Industrial IoT," *Journal of Industrial Information Integration*, vol. 22, p. 100196, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2452414X20300716

[9] W. Hofmann and F. Branding, "Implementation of an IoT- and Cloud-based Digital Twin for Real-Time Decision Support in Port Operations," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 2104–2109, 2019, 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896319315009

[10] A. Saad, S. Faddel, and O. Mohammed, "IoT-Based Digital Twin for Energy Cyber-Physical Systems: Design and Implementation," *Energies*, vol. 13, no. 18, 2020. [Online]. Available: https://www.mdpi.com/1996-1073/13/18/4762

[11] A. M. Madni, C. C. Madni, and S. D. Lucero, "Leveraging Digital Twin Technology in Model-Based Systems Engineering," *Systems*, vol. 7, no. 1, 2019. [Online]. Available: https://www.mdpi.com/2079-8954/7/1/7

[12] A. J. Zakrajsek and S. Mall, "The Development and Use of a Digital Twin Model for Tire Touchdown Health Monitoring," in *58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2017, p. 0863.

[13] D. Liu, T. Abdelzaher, T. Wang, Y. Hu, J. Li, S. Liu, M. Caesar, D. Kalasapura, J. Bhattacharyya, N. Srour, J. Kim, G. Wang, G. Kimberly, and S. Yao, "IoBT-OS: Optimizing the Sensing-to-Decision Loop for the Internet of Battlefield Things," in *2022 International Conference on Computer Communications and Networks (ICCCN)*, 2022, pp. 1–10.

[14] A. Somov, C. Dupont, and R. Giaffreda, "Supporting Smart-City Mobility with Cognitive Internet of Things," 01 2013, pp. 1–10.

[15] D. Kyriazis, T. Varvarigou, D. White, A. Rossi, and J. Cooper, "Sustainable Smart City IoT Applications: Heat and Electricity Management Eco-conscious Cruise Control for Public Transportation," in *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2013, pp. 1–5.

[16] P. Hank, S. Müller, O. Vermesan, and J. V. den Keybus, "Automotive Ethernet: In-Vehicle Networking and Smart Mobility," *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1735–1739, 2013.

[17] N. Bressan, L. Bazzaco, N. Bui, P. Casari, L. Vangelista, and M. Zorzi, "The Deployment of a Smart Monitoring System Using Wireless Sensor and Actuator Networks," 11 2010, pp. 49 – 54.

[18] A.-H. Mohsenian-Rad, V. W. S. Wong, J. Jatskevich, R. Schober, and A. Leon-Garcia, "Autonomous Demand-Side Management Based on Game-Theoretic Energy Consumption Scheduling for the Future Smart Grid," *IEEE Transactions on Smart Grid*, vol. 1, no. 3, pp. 320–331, 2010.

[19] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A Software Defined Networking Architecture for the Internet-of-Things," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–9.

[20] M. Wang, G. Zhang, C. Zhang, J. Zhang, and C. Li, "An IoT-based Appliance Control Cystem for Smart Homes," *2013 Fourth International Conference on Intelligent Control and Information Processing (ICICIP)*, pp. 744–747, 2013.

[21] R. Piyare, "Internet of Things: Ubiquitous Home Control and Monitoring System using Android based Smart Phone," *International Journal of Internet of Things*, vol. 2, pp. 5–11, 09 2013.

[22] S. Kumar, "Ubiquitous Smart Home System Using Android Application," *ArXiv*, vol. abs/1402.2114, 2014.

[23] S. Fang, L. Xu, Y. Zhu, J. Ahati, H. Pei, J. Yan, and Z. Liu, "An Integrated System for Regional Environmental Monitoring and Management Based on Internet of Things," *Industrial Informatics, IEEE Transactions on*, vol. 10, pp. 1596–1605, 05 2014.

[24] G. Yang, L. Xie, M. Mäntysalo, X. Zhou, Z. Pang, L. Xu, S. Kao-Walter, Q. Chen, and L.-R. Zheng, "A Health-IoT Platform Based on the Integration of Intelligent Packaging, Unobtrusive Bio-Sensor and Intelligent Medicine Box," *IEEE Transactions on Industrial Informatics*, vol. 10, pp. 1–1, 11 2014.

[25] P. J. Reaidy, A. Gunasekaran, and A. Spalanzani, "Bottom-Up Approach based on Internet of things for Order Fulfillment in a Collaborative Warehousing Environment," 2015.

[26] S. Li, "Application of the Internet of Things Technology in Precision Agriculture IRrigation Systems," ser. CSSS '12. USA: IEEE Computer Society, 2012. [Online]. Available: https://doi.org/10.1109/CSSS.2012.256 p. 1009–1013.

[27] A. Khanna and S. Kaur, "Internet of Things (IoT), Applications and Challenges: A Comprehensive Review," *Wireless Personal Communications*, vol. 114, no. 2, pp. 1687–1762, May 2020. [Online]. Available: https://doi.org/10.1007/s11277-020-07446-4

[28] M. Singh, E. Fuenmayor, E. P. Hinchy, Y. Qiao, N. Murray, and D. Devine, "Digital Twin: Origin to Future," *Applied System Innovation*, vol. 4, no. 2, p. 36, 2021.

[29] S. A. Ferguson, "Apollo 13: The first digital twin," Mar 2021. [Online]. Available: https://blogs.sw.siemens.com/simcenter/apollo-13-the-first-digital-twin/

[30] F. Biesinger and M. Weyrich, "The Facets of Digital Twins in Production and the Automotive Industry," in *2019 23rd International Conference on Mechatronics Technology (ICMT)*, 2019, pp. 1–6.

[31] A. F. Mendi, T. Erol, and D. Doğan, "Digital Twin in the Military Field," *IEEE Internet Computing*, vol. 26, no. 5, pp. 33–40, 2022.

[32] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital Twin in Manufacturing: A Categorical Literature Review and Classification," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–1022, 2018, 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896318316021

[33] B. Bjornsson, C. Borrebaeck, N. Elander, T. Gasslander, D. Gawel, M. Gustafsson, R. Jörnsten, E. J. Lee, X. Li, S. Lilja, D. Martínez-Enguita, A. Matussek, P. Sandström, S. Schäfer, M. Stenmarker, X. Sun, O. Sysoev, H. Zhang, and M. Benson, "Digital twins To Personalize Medicine," *Genome Medicine*, vol. 12, 12 2019.

[34] E. Shahat, C. T. Hyun, and C. Yeom, "City Digital Twin Potentials: A Review and Research Agenda," *Sustainability*, vol. 13, no. 6, 2021. [Online]. Available: https://www.mdpi.com/2071-1050/13/6/3386

[35] J.-F. Uhlenkamp, K. Hribernik, S. Wellsandt, and K.-D. Thoben, "Digital Twin Applications : A First Systemization of their Dimensions," in *2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, 2019, pp. 1–8.

[36] S. Liu, X. V. Wang, and L. Wang, "Digital Twin-enabled Advance Execution for Human-Robot Collaborative Assembly," *CIRP Annals*, vol. 71, no. 1, pp. 25–28, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000785062200018X

[37] C. Verdouw, B. Tekinerdogan, A. Beulens, and S. Wolfert, "Digital Twins in Smart Farming," *Agricultural Systems*, vol. 189, p. 103046, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0308521X20309070

[38] A. K. Ghosh, A. S. Ullah, R. Teti, and A. Kubo, "Developing Sensor Signal-based Digital Twins for Intelligent Machine Tools," *Journal of Industrial Information Integration*, vol. 24, p. 100242, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2452414X21000418

[39] A. Sharma, E. Kosasih, J. Zhang, A. Brintrup, and A. Calinescu, "Digital Twins: State of the Art Theory and Practice, Challenges, and Open Research Questions," *Journal of Industrial Information Integration*, vol. 30, p. 100383, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2452414X22000516

[40] M. Picone, M. Mamei, and F. Zambonelli, "WLDT: A General Purpose Library to Build IoT Digital Twins," *SoftwareX*, vol. 13, p. 100661, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711021000066

[41] Z. Lei, E. Sun, S. Chen, J. Wu, and W. Shen, "A Novel Hybrid-Copy Algorithm for Live Migration of Virtual Machine," *Future Internet*, vol. 9, no. 3, 2017. [Online]. Available: https://www.mdpi.com/1999-5903/9/3/37

[42] F. Tao, F. Sui, A. Liu, Q. Qi, M. Zhang, B. Song, Z. Guo, S. C.-Y. Lu, and A. Y. C. Nee, "Digital Twin-Driven Product Design Framework," *International Journal of Production Research*, vol. 57, no. 12, pp. 3935–3953, 2019. [Online]. Available: https://doi.org/10.1080/00207543.2018.1443229

[43] K. Sivalingam, M. Sepulveda, M. Spring, and P. Davies, "A review and methodology development for remaining useful life prediction of offshore fixed and floating wind turbine power converter with digital twin technology perspective," in *2018 2nd International Conference on Green Energy and Applications (ICGEA)*, 2018, pp. 197–204.

[44] W. Luo, T. Hu, C. Zhang, and Y. Wei, "Digital Twin for CNC Machine Tool: Modeling and Using Strategy," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, 03 2019.

[45] J. Wilhelm, T. Beinke, and M. Freitag, "Improving Human-Machine Interaction with a Digital Twin: Adaptive Automation in Container Unloading," 02 2020.

[46] Y. Xu, Y. Sun, X. Liu, and Y. Zheng, "A Digital-Twin-Assisted Fault Diagnosis Using Deep Transfer Learning," *IEEE Access*, vol. 7, pp. 19 990–19 999, 2019.

[47] P. Jain, J. Poon, J. P. Singh, C. Spanos, S. R. Sanders, and S. K. Panda, "A Digital Twin Approach for Fault Diagnosis in Distributed Photovoltaic Systems," *IEEE Transactions on Power Electronics*, vol. 35, no. 1, pp. 940–956, 2020.

[48] J. Guo, N. Zhao, L. Sun, and Z. Saipeng, "Modular-based Flexible Digital Twin for Factory Design," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, 03 2019.

[49] D. Iglesias, P. Bunting, S. Esquembri, J. Hollocombe, S. Silburn, L. Vitton-Mea, I. Balboa, A. Huber, G. Matthews, V. Riccardo, F. Rimini, and D. Valcarcel, "Digital Twin Applications for the JET Divertor," *Fusion Engineering and Design*, vol. 125, pp. 71–76, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920379617308748

[50] S. Haag and R. Anderl, "Digital Twin – Proof of Concept," *Manufacturing Letters*, vol. 15, pp. 64–66, 2018, industry 4.0 and Smart Manufacturing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2213846318300208

[51] J. Leng, H. Zhang, D. Yan, Q. Liu, X. Chen, and D. Zhang, "Digital Twin-Driven Manufacturing Cyber-Physical System for Parallel Controlling of Smart Workshop," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, p. 1155–1166, 03 2019.

[52] F. Akbarian, E. Fitzgerald, and M. Kihl, "Synchronization in Digital Twins for Industrial Control Systems," 2020. [Online]. Available: https://arxiv.org/abs/2006.03447

[53] Y. Han, D. Niyato, C. Leung, D. I. Kim, K. Zhu, S. Feng, S. X. Shen, and C. Miao, "A Dynamic Hierarchical Framework for IoT-assisted Digital Twin Synchronization in the Metaverse," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[54] H. Zipper, "Real-Time-Capable Synchronization of Digital Twins," *IFAC-PapersOnLine*, vol. 54, no. 4, pp. 147–152, 2021, 4th IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control CESCIT 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896321014270

[55] J. Beuchert, F. Solowjow, S. Trimpe, and T. Seel, "Overcoming bandwidth limitations in wireless sensor networks by exploitation of cyclic signal patterns: An event-triggered learning approach," *Sensors (Basel)*, vol. 20, no. 1, Jan. 2020.

[56] Y. S. Suh, "Send-On-Delta Sensor Data Transmission With A Linear Predictor," *Sensors*, vol. 7, no. 4, pp. 537–547, 2007. [Online]. Available: https://www.mdpi.com/1424-8220/7/4/537

[57] Y. Mo, R. Ambrosino, and B. Sinopoli, "Sensor Selection Strategies for State Estimation in Energy Constrained Wireless Sensor Networks," *Automatica*, vol. 47, no. 7, p. 1330–1338, jul 2011. [Online]. Available: https://doi.org/10.1016/j.automatica.2011.02.001

[58] G. Battistelli, A. Benavoli, and L. Chisci, "State Estimation with Remote Sensors and Intermittent Transmissions," *Systems Control Letters*, vol. 61, no. 1, pp. 155–164, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016769111100260X

[59] L. Cheng, S. Hailes, Z. Cheng, F.-Y. Fan, D. Hang, and Y. Yang, "Compressing Inertial Motion Data in Wireless Sensing Systems — An Initial Experiment," in *2008 5th International Summer School and Symposium on Medical Devices and Biosensors*, 2008, pp. 293–296.

[60] J. Sztipanovits, T. Bapty, S. Neema, L. Howard, and E. Jackson, "OpenMETA: A Model-and Component-based Design Tool Chain for Cyber-Physical Systems," in *From programs to systems. The systems perspective in computing.* Springer, 2014, pp. 235–248.

[61] S. Russell and T. Abdelzaher, "The Internet of Battlefield Things: The Next Generation of Command, Control, Communications and Intelligence (C3I) Decision-Making," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM).* IEEE, 2018, pp. 737–742.

[62] "Raspberry Shake," https://raspberryshake.org/.

[63] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "DeepSense: A Unified Deep Learning Framework for Time-Series Mobile Sensing Data Processing," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2017. [Online]. Available: https://doi.org/10.1145/3038912.3052577 p. 351–360.

[64] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[65] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, Architecture, and Uses in the Wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074