PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

A CYBER-PHYSICAL SYSTEMS APPROACH TO COLLABORATIVE INTERSECTION MANAGEMENT AND CONTROL

JOSÉ ANTONIO GUZMÁN GÓMEZ

Thesis submitted to the Office of Graduate Studies
in partial fulfillment of the requirements for the degree of
Doctor of Science in Engineering

Advisor:
FELIPE NÚÑEZ

Santiago de Chile, March 2023

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

A CYBER-PHYSICAL SYSTEMS APPROACH TO COLLABORATIVE INTERSECTION MANAGEMENT AND CONTROL

JOSÉ ANTONIO GUZMÁN GÓMEZ

Members of the Committee:

FELIPE NÚÑEZ
SANDRA CÉPEDES
HANS LÖBEL
FELIPE DELGADO
BRENDAN MORRIS
JUAN DE DIOS ORTÚZAR

Thesis submitted to the Office of Graduate Studies in partial fulfillment of the requirements for the Degree Doctor in Engineering Sciences

Santiago de Chile, March, 2023

Gratefully to my parents,
grandparents and sister

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Prof. Felipe Núñez, for his invaluable advice, constant support, and patience during my PhD study. His vast knowledge and experience have guided me throughout my academic research.

I would like to thank Prof. Brendan T. Morris from the University of Nevada, La Vegas, for agreeing to be the supervisor of my PhD internship during the uncertain times of the COVID-19 pandemic.

I would like to thank my family and friends for always encouraging me in the most complicated and stressful moments of my PhD study.

A would like to thank Dirección de Postgrado and Vicerrectoria de Investigación at Pontificia Universidad Católica de Chile, for supporting my doctoral research.

Finally, I thank the Agencia Nacional de Investigación y Desarrollo (ANID) for supporting this research through the ANID-PFCHA/Doctorado Nacional/2019-21190519 and the ANID PIA ACT192013 grants.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

As urban population increases steadily, traffic congestion has become a major source of discomfort and economic losses in urban environments. In this context, the concept of intelligent transportation systems takes relevance as a way of optimizing traffic control with the use of technology. This work proposes a cyber-physical systems approach to collaborative urban traffic control. Specifically, a three-layer architecture is proposed to address the intersection management problem, with a physical layer formed by traffic detectors and the traffic signal actuator, a cyber layer in charge of performing data processing, communication with neighbors and traffic signal actuation, and a cloudlet layer capable of implementing high level applications.

A first implementation in a pseudo-real environment, using a designed fuzzy-expert controller and a novel timed-Preti-net based actuator, shows that the proposed system is capable of handling the communication and processing loads, while improving traffic performance with respect to classical solutions, outperforming timed, Webster and coordinated scheduling methods in pilot-scale tests.

To validate the architecture in a real environment, the implementation of a real-time traffic prediction application in the Las Vegas urban area is presented, which is built as a cloudlet application with real-time data streaming from field sensors and deep-learning-based traffic predictors. Implementation results show the feasibility of doing traffic prediction in real time with the current technology and the usefulness of periodic retraining to maintain prediction accuracy.

Finally, the thesis concludes by proposing a distributed control scheme based on reinforcement learning (RL) that exploits the data-driven nature of the problem, the cooperation between intersections, and the modularity of the proposed architecture. Specifically, a RL controller is synthesized, which manipulates traffic signals using information from

neighboring intersections in the form of an embedding obtained from a traffic prediction application. Simulation results using SUMO show that the proposed scheme outperforms classical techniques in terms of waiting time and other key performance indices.

Keywords: Intelligent Transportation Systems, Cyber-physical Systems, Intersection management, Traffic prediction, Deep learning, Reinforcement learning, Timed-petri-net.

RESUMEN

A medida que la población urbana aumenta, la congestión vehicular se ha convertido en una fuente importante de molestias y pérdidas económicas en las zonas urbanas. En este contexto, el concepto de Sistemas Inteligentes de Transporte toma relevancia como opción para optimizar el control del tráfico con el uso de la tecnología. Este trabajo propone un enfoque de sistemas ciberfísicos para el control colaborativo del tráfico urbano. En concreto, se propone una arquitectura de tres capas para abordar el problema de la gestión de intersecciones, con una capa física que alberga los detectores de tráfico y los actuadores de las señales de tráfico, una capa cibernética encargada de realizar el procesamiento de datos, la comunicación con los vecinos y la manipulación de las señales de tráfico, y una capa cloudlet capaz de implementar aplicaciones de alto nivel.

Una primera implementación en un entorno pseudo-real, utilizando un controlador fuzzy-experto diseñado y un novedoso actuador basado en Petri-nets temporizadas, muestran que el sistema propuesto es capaz de manejar las cargas de comunicación y procesamiento, a la vez que mejora el rendimiento del tráfico con respecto a las soluciones clásicas de manejo de señales de tráfico, superando a los métodos de control temporizado, Webster y coordinado en las pruebas a escala piloto.

Para validar la arquitectura en un entorno real, una aplicación de predicción de tráfico en tiempo real en el área urbana de Las Vegas es presentada, construida como una aplicación cloudlet con flujo de datos en tiempo real desde sensores de la infraestructura y predictores de tráfico basados en modelos de aprendizaje profundo. Los resultados de la implementación muestran la viabilidad de hacer predicciones de tráfico en tiempo real con la tecnología actual y la utilidad del reentrenamiento periódico para mantener la precisión de la predicción.

Finalmente, este trabajo propone un esquema de control distribuido basado en el aprendizaje reforzado (RL) que aprovecha la naturaleza del problema basada en datos, la cooperación entre intersecciones y la modularidad de la arquitectura propuesta. En concreto, se sintetiza un controlador RL que manipula los semáforos utilizando la información de las intersecciones vecinas en forma de un embebido obtenido de una aplicación de predicción de tráfico. Los resultados de la simulación con SUMO muestran que el esquema propuesto supera a las técnicas clásicas en términos de tiempo de espera y otros índices clave de rendimiento.

Palabras Claves: Sistemas Inteligentes de Transporte, Sistemas ciberfísicos, Manejo de intersecciones, Predicción de tráfico, Aprendizaje profundo, Aprendizaje por refuerzo, Petri-nets temporizadas.

1. INTRODUCTION

1.1. Motivation

Urban population has experienced an exponential growth in recent years, a trend that seems to remain firm in the upcoming decades. It is estimated that by the year 2050 the world population will reach 9.8 billion inhabitants, two thirds of which will live in large cities, representing an increase of 36% from today's situation (United Nations, 2018). This new reality will inevitably render mobility and transportation services offered in urban areas inefficient and insufficient, incurring serious consequences for citizens.

In this scenario, vehicle congestion is one of the most serious problems, since it causes extreme economic losses ($88 billion in the USA in 2019) (Reed, 2020) derived from unnecessary expenditure of fuel, deterioration of vehicles and decrease in the productivity of the city. In addition, it also affects the health of the inhabitants due to the generation of air pollution, delays in emergency services, and nervous diseases (Cookson, 2018). Although in 2020 there was a drastic decrease in vehicular traffic globally, caused mainly by the worldwide lockdowns imposed due to the COVID-19 pandemic, recent data shows that the previous upward trend in traffic congestion could quickly resume. Indeed, in 2021, traffic congestion in the U.S. costed drivers over $53 billion, a 41% increase from the previous year (Pishue, 2021). Finding efficient ways to deal with traffic congestion is imperative.

The most widespread solution to reduce urban congestion is the use of Urban Traffic Control (UTC) systems for efficient intersection management, which represent the core of modern Intelligent Transportation Systems (ITS) (X. Zhang & Riedel, 2017) due to its high effectiveness, low cost and low space requirements compared to structural solutions (Hamilton et al., 2013). A UTC system may implement a variety of control strategies on traffic signals for intersection management, from the simplest fixed time plans, to more complex and effective alternatives such as coordination at different intersections and real-time actuation based on street detector data.

Currently, the vast majority of UTC systems adjust the split (time that a light is in *green* within a phase), offset and cycle time of traffic lights to optimize traffic, but differ in the computational architecture and the strategies used to optimize these variables. SCOOT (Robertson & Bretherton, 1991), for example, is the most common UTC system, which is coordinated from a central computer that uses live traffic data to determine signalling times. By comparison, SCATS (Sims & Dobinson, 1980) is a distributed approach that applies different fixed time plans (created based on historical data) depending on the current scenario. Most advanced UTC systems use hierarchical architectures to distribute the work UTOPIA (Mauro & Taranto, 1990), RHODES (Mirchandani & Head, 2001), MOTION (Brilon & Wietholt, 2013), allowing the use of more complex strategies, as online estimation and predictive models (Hamilton et al., 2013).

A common factor to all UTC systems is that a better understanding of road and city conditions drastically improves performance. In this sense, on the one hand we have that recent advances in sensor and communication technologies have contributed to the availability of massive traffic data, from which road and city conditions can be determined in real time; it is not surprising that current trends in intersection management use more information together with intelligent control algorithms to define the system actions, looking to automate processes and achieve real-time response to traffic conditions (Chen & Englund, 2016) (UTMC Ltd, 2009). On the other hand, collaborative strategies appear to be a promising approach for extracting information from the field, since the exchange of data between system elements contributes to create a richer picture of the state of the environment (Chen & Englund, 2016).

The Internet of Things (IoT) (Gubbi et al., 2013) plays an important role in this trend, acting as a large-scale data provider platform (Zanella et al., 2014) that allows real-time data gathering and communication to describe the performance of the different services in a city. As a natural consequence, data-driven algorithms, especially those based on deep neural networks (DNNs), have gain relevance in the traffic management context, allowing

the use of the collected data to implement high-level applications, such as traffic prediction or automatic controllers that act in real-time on traffic signals. In this scenario, cyber-physical systems (CPSs) (Harrison et al., 2016), conceived as highly distributed connected systems with the ability of actuating on their surrounding physical environment and the capacity to perform highly demanded tasks, appear as the natural tool for implementing smart services capable of autonomously and efficiently control parts of the urban environment. CPSs have shown great success in industry (Langarica et al., 2020; Núñez et al., 2020), healthcare (Y. Zhang et al., 2017) and energy (Ye et al., 2015), which makes them an appealing alternative for tackling the intersection management problem using real-time data.

This work looks to set the basis for using a CPSs paradigm to create collaborative solutions to the intersection management and control problem. Supported by the natural scalability, modularity and horizontal integration of CPSs, our proposal enables implementing a variety of services with local (within a neighborhood or even at a single intersection) and global scope. Moreover, our proposed approach allows decision making at different time-scales within the same system architecture. Furthermore, the flexible CPS architecture promotes rapid prototyping and testing of data-driven services by taking advantage of the standardized information models and the processing capabilities deployed at all levels: field, network (fog) and cloud.

1.2. Hypotheses, Objectives and Contributions

The driving hypothesis of this work is that the CPSs paradigm is an appropriate tool for tackling the intersection management and control problem, since it allows materializing a real-time data-driven decision making, from local to global scopes. A secondary hypothesis, which motivates the search for a paradigm that enables real-time data-driven decision making, is that decision making based on real-time traffic data, particularly a collaborative scheme, outperforms classical intersection management techniques in terms of traffic congestion metrics, such as the average vehicle waiting time.

Consequently, the main objective of this thesis is to propose a CPSs scheme for collaborative intersection management (CIM), and to validate it by deploying and evaluating a series of concrete data-driven applications that have the potential of improving traffic conditions in an urban scenario.

To achieve the main objective, the following specific objectives are in order:

(i) To design a CPS architecture for collaborative traffic intersection management that fulfills the functional requirements for real-world implementations.
(ii) To validate the CPS architecture by implementing a realistic pilot instance for functional and performance testing.
(iii) To design, implement in a real environment, and validate in terms of proper performance metrics, a high-level broad-scope data-driven application that can be deployed on the designed architecture.
(iv) To design, implement, and validate in terms of proper performance metrics, a local-scope CIM data-driven application.

In the achievement of the aforementioned objectives, this thesis delivers the following concrete contributions to the scientific community:

- A scalable, robust, secure and safe traffic-oriented three-layer CPS architecture that allows using multiple data sources and implementing intelligent collaborative real-time decision making in urban scenarios.
- A novel timed-Petri net-based traffic signal actuator that enhances existing Petri net-based efforts to allow secure real-time actuation and event reaction, hence enabling the use of data-driven decision making focused on modifying the behavior of traffic signals.
- A proof of concept of a real-time data-driven traffic prediction application implemented using real detectors from Las Vegas I-15 Freeway and based on two novel DNN prediction models with error recurrence.

- A novel distributed proximal policy optimization (PPO) RL controller for cooperative intersection traffic control, which works in tandem with a graph neural network (GNN) traffic prediction model that encodes the information of intersections and feed the PPO RL controller.

1.3. Organization

The rest of this thesis is organized as follows[1]: In Chapter 2 the proposed CPS architecture for CIM is described in detail and the laboratory implementation of a concrete instance is presented. Chapter 3 is focused on the design and real-world implementation of an application for real-time traffic prediction on the Las Vegas I-15 Freeway. In Chapter 4 the distributed PPO RL-based control scheme for cooperative intersection traffic control is introduced and evaluated. Finally, in Chapter 5 concluding remarks and directions for future research are stated.

[1]This thesis follows the "3-paper format" of the Pontificia Universidad Católica de Chile, in which the thesis is the compendium of three research articles developed by the student. Although redundant material between chapters has been minimized, some redundancy is unavoidable since removing material affects the flow of the articles in the compendium.

2. PROPOSED SYSTEM ARCHITECTURE

This Chapter deals with the proposed CPS architecture for intersection management and control[2]. Therefore, it contributes to the achievement of specific objectives (i) and (ii).

2.1. Context

Current research on UTC systems focuses on two core methods: intelligent-vehicle-based and intelligent-traffic-signal-based.

Intelligent-vehicle-based methods implement control strategies for Automated Vehicles (AVs) adopting vehicle-to-vehicle (V2V) or infrastructure-to-vehicle (I2V) communication technologies (Lin et al., 2013). In this context, a promising solution to obtain fuel-efficient and safe driving are time slot allocation strategies, where AVs communicate their route, position and velocity to an external agent that applies an algorithm to assign a time slot to each vehicle for intersection crossing (Jin et al., 2012; Bichiou & Rakha, 2018). Alternatively, in safe trajectory prediction strategies, the external agent uses car information to predict safe and unsafe trajectories, and sends acceleration and deceleration commands to the AVs to ensure safety and efficiency (Abdelhameed et al., 2014a, 2014b; Buzachis et al., 2019).

On the other hand, intelligent-traffic-signal-based methods improve the management of existing traffic signals, which makes their real implementation feasible in a shorter time period. Most of these methods focus on designing better control strategies as: agent-based approaches, with intersections modeled as agents following rules stored in their knowledge base (Guerrero-Ibáñez et al., 2010), back-pressure control approaches, which implement a distributed algorithm to compute control actions based on queue lengths and can achieve probably maximum stability (Gregoire et al., 2015), and model predictive

[2]The material in this chapter is published in "Guzmán, J., & Núñez, F. (2021). A cyber-physical systems approach to collaborative intersection management and control. IEEE Access, 9, 99617-99632." (Guzmán & Núñez, 2021).

control approaches, which use street network simulations to predict future states of the system and make traffic lights adjustments accordingly (Lin et al., 2013; Zhou et al., 2017).

A related area of research, important for UTC systems, focuses on real-time automated control actions and reaction to unexpected events. In this area, Petri nets (PNs) have arisen as a standard tool because of their capacity for synchronization and coordination of distributed timed discrete processes, fast response, and safe and simple implementation. In Qi et al. (2016, 2018), a Deterministic and Stochastic Petri net (DSPN) model of traffic signals for accidents warning and management is proposed, introducing two new displays: *warning* and *emergency*. The design was tested over a rectangular grid in a simulated environment, where the model performs well in response to an incident under certain conditions including timely termination of their use after clearance. In Huang et al. (2015), a timed Petri net (TPN) model of a traffic signal that reacts by giving green light as soon as possible to emergency vehicles (EV) is designed. Six types of EV arrival events were defined for the system to attend to. List and Cetin (2004) introduced a modular TPN model for traffic signal safe and real-time control. The model consists in two modules, one for display transitions and other to control phase transitions. The main advantage is that it allows transitioning between any phases and applying timed and actuated control strategies to change the split of the phases, while ensuring safety by denying combinations of signals that can cause collisions. Most of these solutions are very specific to the scenario they are tested on, requiring major modifications to implement them over real intersections, which are heterogeneous in terms of structure.

A promising strategy that takes advantage of new communication technologies is CIM, where several system members, such as Vehicle On Board Units, intersections and other transportation and city services, share information to make collaborative decisions (Chen & Englund, 2016). In Hirankitti and Krohkaew (2007) a collaborative approach for intelligent traffic control is proposed, where each intersection is modeled as an agent that communicates with its neighbors and controls its traffic signals following a set of rules involving its own traffic condition, along with the *downstream space availability* of its

neighbors. This solution outperforms fixed time and actuated controls. In Xu et al. (2019) a cooperative method for traffic signal optimization and vehicle speed control is proposed, where the intersection acquires the speed and position information from approaching vehicles to calculate the optimal traffic signal times and plans the vehicles' arrival times with the aim of decreasing the total travel time of all vehicles. By sending the planned arrival times to the vehicles, they can control locally their systems to arrive at the deadline. This solution was shown to be more effective than actuated control and vehicle control approaches with no intersection communication. However, most of the tests are simulated over very controlled scenarios, without being exposed to real communication, data transmission issues and real-time restrictions.

2.2. A Hierarchical Distributed CPS Architecture to CIM

To design an intelligent intersection management system, a hierarchical distributed CPS approach is taken. To this end, each intersection is modeled as a two layer cyber-physical object (CPO), with a physical layer that includes the physical environment and its instrumentation, and a cyber layer, which is divided in three modules that cover: i) congestion model, ii) traffic signals model, and iii) communication with other CPOs and intelligent control. Each CPO instance communicates with neighboring CPOs to obtain information of the congestion of their associated intersections. In this way, each CPO can calculate the split of its local traffic signals **collaboratively**, by using its own congestion and that of its neighboring intersections. Finally, each CPO from a neighborhood communicates its state to a layer of superior intelligence, called the "cloudlet" layer. The cloudlet is a high-performance computational layer, which allows the execution of more complex operations with the collected data. The cloudlet has cloud capabilities, but is located near the neighborhood where control actions are being applied. Unlike the cloud, access to the cloudlet is achieved via a permanent communication link to ensure low latency, and thus achieve a real-time response. This layer provides a consolidation point for applications to

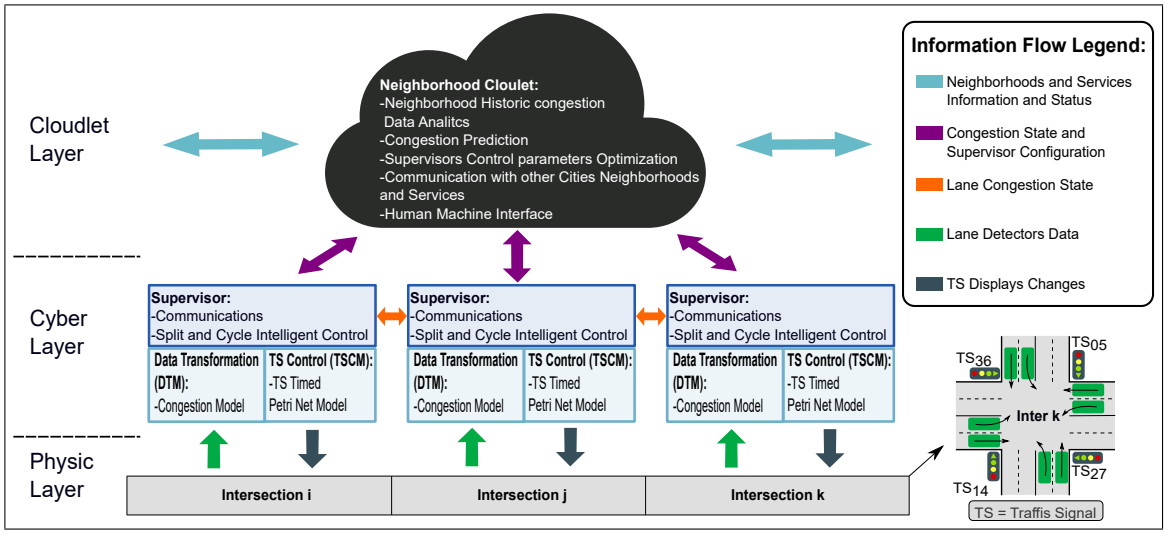


Figure 2.1. Cyber-physical system concept of the proposed intersection management system. Physical Layer: intersection detectors and actuators; Modular Cyber Layer: Data Transformation Module, Traffic Signal Control Module and Supervisor Module; Cloudlet Layer: high level applications

optimize the parameters of the control strategy, communicates with other city neighborhoods and services, predicts events, presents a global perspective of the network state, and presents statistical information to governmental entities, among others.

The proposed CPS architecture is illustrated in Fig. 2.1, and detailed in the following subsections.

2.2.1. Physical Layer: Detectors and Actuators

The physical layer of an intersection is composed by all types of detectors that may be present for data collection, like inductive-loops, cameras, lidar, radar, GPS (Shirazi & Morris, 2017), or Roadside Units (RSUs) that receive status information from vehicles' On-Board Units (OBUs). It also contains traffic signals that are used as actuators to control traffic at the intersection. There is no restriction on the types of detectors that can be used, but this decision impacts the quality of information that can be extracted, and the design of part of the cyber layer.

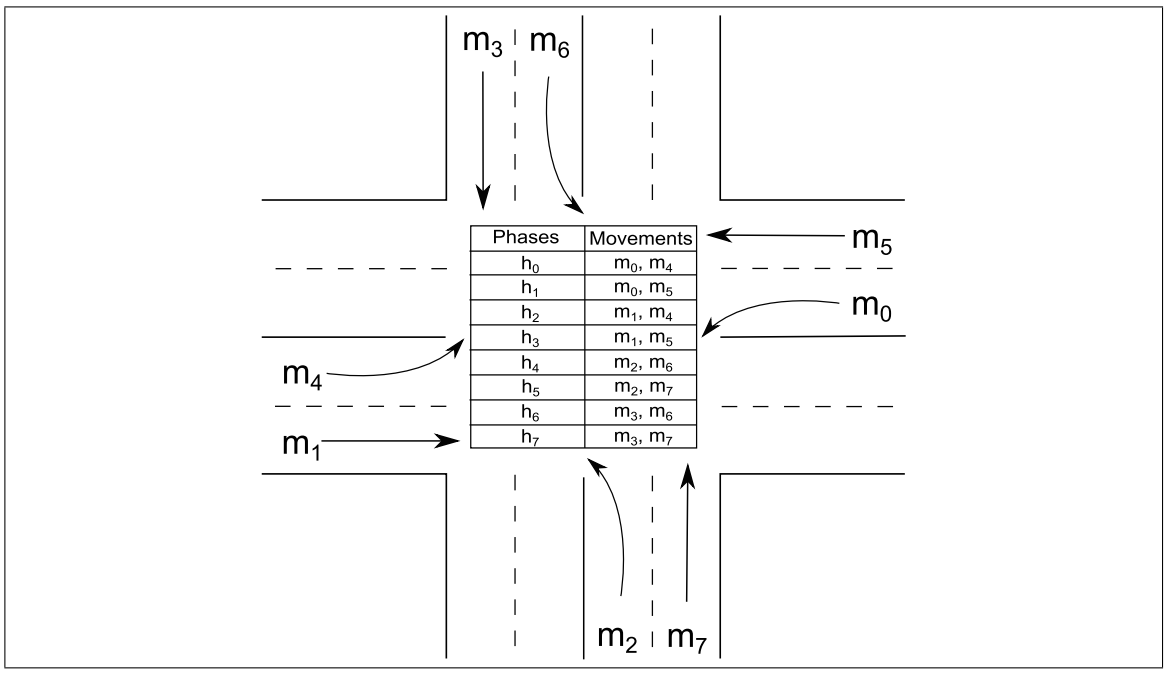


Figure 2.2. Intersection base model.

As a premise for the physical layer in this architecture, intersections must be fully instrumented, so they can obtain traffic data in real-time and manipulate traffic signal displays when required. Intersections can work sub-optimally if any of their neighbors is not instrumented.

2.2.2. Cyber Layer: Cyber Intersection Model

To develop the cyber layer, the intersection model presented in List and Cetin (2004) (Fig. 2.2) is considered, in which the concepts of movements, phases and cycles are presented. A *movement* m_i is a flow of cars in a specific direction, and a *phase* h_j is a set of movements that flow simultaneously, within the time interval that this phase is active. As it can be seen in Fig. 2.2, there are eight possible movements ($i \in \{0, 1, \ldots, 7\}$), where the even: m_0, m_2, m_4 and m_6 are left turns, and the odd: m_1, m_3, m_5 and m_7 are through-right combinations. For safety and efficiency, the model defines eight phases ($j \in \{0, 1, \ldots, 7\}$) as pairs of movements that flow without interrupting each other. Only

these combinations are allowed. Finally, a cycle l_k is a defined sequence of phases that is implemented in the traffic light in a cyclic manner.

The cyber layer is composed by three modules that communicate and interact to manage the intersection efficiently.

2.2.2.1. Data Transformation Module (DTM)

One of the advantages offered by the proposed architecture is the processing capacity located near the process, following the fog computing paradigm in CPSs (Tang et al., 2017), which allows processing the detector's data without compromising real-time performance. To this end, a model that concisely represents the congestion state of the intersection, by synthesizing the relevant information from the collected data, was defined. This model operates in the DTM, and computes a real number $c_i \in [0, 100]$ that represents the congestion state at each of the movements of the intersection. The congestion state can be calculated by applying any type of algorithm that transforms the data collected by the physical layer detectors into the congestion state values. This algorithm is applied when requested by another cyber layer module.

One of the clear advantages of this strategy is that it makes the system detector agnostic, since it is a matter of designing and implementing a congestion model corresponding to the data type, thus dealing with the well-known heterogeneity problem in CPSs (Marwedel & Engel, 2016). It also allows the system to easily integrate new sources of information, as new detectors or new paradigms like Vehicle to Infrastructure communication (V2I) with information received from Vehicles On-Board Units (OBU) or from Automated Vehicles (AV) directly. In addition, the system can be improved by using different algorithms for congestion modeling, without changing the rest of the system.

2.2.2.2. Traffic Signal Control Module (TSCM)

The TSCM is modeled as a TPN in charge of managing the ON/OFF commands of the traffic signals, performing the cyclic phase change, modifying the signal cycle when an event occurs, and communicating the traffic signals state.

To understand this module, TPNs must be formally introduced. PNs are a graphical and mathematical modeling tool beneficial for describing information processing systems that are concurrent, asynchronous, distributed, parallel, non-deterministic, and/or stochastic (Murata, 1989). PN graphs are composed of four elements: places, that usually represent states or processes of the system; transitions, that represent events; tokens, that represent objects or a pointer to a current active process; and directed arcs, which indicate the flow direction of tokens throughout the net. Places connect to transitions through input arcs and transitions connect to places through output arcs. Tokens are located in places.

In their basic form, PNs are not capable of handling dynamic systems where time is a fundamental variable. TPNs address this issue. In a TPN, two time values are defined for each transition, α^s and β^s, where α^s, also known as the *static earliest firing time* (static EFT), is the minimum time the transition must wait after it is enabled and before it is fired, and β^s, called the *static latest firing time* (static LFT), which is the maximum time the transition can wait before firing if it is still enabled. An unweighted TPN is a 6-tuple:

$$N = (P, T, I, O, M_0, SI), \tag{2.1}$$

where $P = \{p_1, p_2, ..., p_m\}$ is a finite set of places, $T = \{t_1, t_2, ..., t_n\}$ is a finite set of transitions, with $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$, $I : T \times P \rightarrow \{0, 1\}$ is an input function defining directed arcs from places to transitions, $O : T \times P \rightarrow \{0, 1\}$ is an output function defining directed arcs from transitions to places, $M_0 : P \rightarrow \mathbb{N}$ is the initial marking function (here $\mathbb{N}$ is the set of nonnegative integers), and $SI : T \rightarrow \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \infty)$ is the static interval (here $\mathbb{Q}^+$ is the set of positive rationals).

An arc directed from place p_j to transition t_i defines p_j as an input place of t_i, implying $I(t_i, p_j) = 1$, while an arc directed from transition t_i to place p_j defines p_j as an output place of t_i, implying $O(t_i, p_j) = 1$. The execution of a PN is controlled by the distribution of tokens. A marking M in a PN is an assignment of tokens to places, which changes during execution. A transition $t_i \in T$ is said to be enabled by marking M at time τ if $\forall p \in P$, $M(p) \geq I(t_i, p)$.

A general form for a state S of a TPN can be defined as pair $S = (M, N)$, where M is a marking; and N is a set with entries of inequalities, each entry of which describes the upper bound and lower bound of the firing time of an enabled transition. The number of entries of N is given in the number of the transitions enabled by marking M. Furthermore, as N has one entry for each transition enabled by a given marking, the number of the entries of N will vary with the behavior of the PN according to the number of transitions enabled by the current marking. In other words, in different states N may have different numbers of entries.

For each transition t_i there is a $SI(t_i) = (\alpha_i^s, \beta_i^s)$ that determines the interval where t_i is firable. Assume that transition t_i becomes enabled at time τ in state $S = (M, N)$. Accordingly, transition t_i is firable from state $S = (M, N)$ at time $\tau + \theta$ if and only if both of the following conditions hold:

(i) t_i is enabled by marking M at time τ.

(ii) the relative firing time θ, relative to the absolute enabling time τ, is not smaller than the EFT of transition t_i and not greater than the smallest of the LFTs of all the transitions enabled by marking M.

Finally, firing t at M yields a new marking

$$M'(P) = M(P) - I(t, p) + O(t, p), \ \forall p \in P. \tag{2.2}$$

These rules allow designing dynamical systems that change their states and actuate on the environment over time, following the "safe direction" defined by the model structure. For more information on TPNs the reader is referred to Wang (1998).

When analyzing the intersection management problem, a TPN is an ideal tool to model traffic signals due to: i) it can synchronize and coordinate distributed timed discrete processes; ii) its fast response, easy implementation and graphic representation; iii) the many tools to simulate the models and guarantee the absence of deadlocks; iv) it can be coded in low level languages like C/C++ (Comlan et al., 2017) and deployed in a micro-controller. This increases reliability since those devices cannot be reconfigured by an outsider attacker; and v) it is able to ensure a safe and proper operation of the intersection by not allowing conflicting movements to have green light simultaneously (Huang et al., 2015), hence ensuring that external manipulations cannot generate an unsafe state.

2.2.2.3. Supervisor

The supervisor is an intelligent module responsible for: i) communicating with neighboring intersections, ii) communicating with the cloudlet node, and iii) interacting with the DTM and the TSCM for the collaborative intelligent management of split times.

A highlight of the proposed strategy is the collaborative management of the intersection by using information from neighbors, which are intersections that share at least one road, to determine the local action. To this end, the supervisor communicates with the DTM module to collect the congestion information of the intersection and creates a congestion state vector. Then, the supervisor sends its state to its neighbors and gathers congestion state from all neighboring intersections. To be protocol and communication interface agnostic, information is added to the data model and the configuration of the intersection in order to allow each intersection to recognize the relative direction of each neighbor, as follows.

```
Data Model {
    id: intersection ID
```

```
    type: TrafficFlowObserved
    dateObserved: date of the observation
    mov_congestion: congestion state vector
}
```

Whenever a message of neighbor state change arrives, the supervisor checks if the movements affected form part of those registered as input or output roads in its configuration and stores the data if the answer is affirmative. This stored information, together with its local congestion state, is used in the split calculation process.

When a new phase is activated, the supervisor communicates the calculated split to the TSCM, which configures the corresponding transition in the TPN.

Finally, the supervisor receives all the traffic signals state changes from the TSCM and delivers the corresponding information to the DTM and the upper layer.

2.2.3. Cloudlet Layer: System Global Intelligence

As mentioned before, to increase the functionalities of the system and to incorporate a global perspective, the cloudlet layer, which collects and digests all the congestion information of a neighborhood, is defined. This approach also makes the system consistent with newer UTC paradigms, where services interact at a high level to achieve efficient management of the city's resources (Hamilton et al., 2013).

This node is in charge of: i) the statistical analysis of the information produced by lower nodes, ii) communicating the state of the global system and events to other services of the city to support collaborative decisions, iii) the optimization of parameters used by control strategies operating at the lower layers, and iv) communicating information from other services, such as weather and emergencies, among others, to the nodes at the lower layers, for consideration in control decisions.

2.2.4. Advantages of the architecture

The advantages of the proposed architecture are:

(i) Intersection modeling as a modular CPO, which allows the continuous improvement of the system and the adoption of new paradigms by updating modules one-by-one. Also, each module has its own contribution: i) abstracting detector data in a congestion variable makes the rest of the system detector agnostic; ii) modeling of traffic signals using a TPN, ensures real time response, reliability, security and safety; iii) supervision of other modules and collaborative calculation of splits.

(ii) A Distributed design which makes the system: i) scalable since each CPO is modeled separately and its design depends only on its physical intersection morphology; ii) robust since each CPO is capable of managing its intersection locally, possibly sub-optimally, in case a neighbor is down; and iii) secure since communication can be implemented over a private local network over which rigorous security measures can be placed.

(iii) A cloudlet that provides a global view for intersection management, and allows implementing high-level applications without compromising real-time response.

2.3. A CPS-based UTC Instance

Based on the proposed architecture, a UTC system is designed with focus on the first two layers. A companion chapter will address cloudlet-level applications. Fuzzy logic is used to model congestion and to calculate the splits.

2.3.1. Physical layer

For this design, we assume that the instrumentation consists of cameras pointing in the direction of the entrance channels of each intersection. This assumption is reasonable given the massive number of cameras installed in urban areas, and the vast amount of

information that can be extracted from them using image processing (Shirazi & Morris, 2017). Hence, the area for information collection is bounded by the range of the camera. The following information is assumed as available in real time.

- number of vehicles: Total number of vehicles in the sensed area.
- occupancy: Percentage of measurement area occupied by vehicles.
- mean speed: Mean speed of vehicles in the sensed area.

It should be noted that cameras are not the only instrument capable of delivering this type of information. Other detectors can also deliver the same set of measurements. The system is detector agnostic, hence, the only requirement is the type of information assumed as available for designing the system.

2.3.2. Cyber Layer

2.3.2.1. DTM

For this particular instance, a fuzzy-expert strategy is proposed for the congestion modeling algorithm defined by the architecture. This is due to the ability of these algorithms to model inherently discrete processes and reduce their stochastic and non-deterministic complexities based on expert knowledge. Three measured variables are fuzzified: number of vehicles, occupancy and mean speed; taking into account the information given in Table 2.1. In this case "max vehicle" was defined as "length of the detector area/average car length" and "max speed" was defined as the max speed of the road. For fuzzification, three fuzzy sets are assigned to each input variable and five fuzzy sets to the congestion output $\in [0, 100]$, all with triangular membership functions. Fig. 2.3 illustrates the fuzzy representation of the occupancy and the defuzzification of congestion.

As with any fuzzy-expert system, inference is done based on IF/THEN rules. The knowledge base was designed based on the idea that the three input variables can be separated in two sets. The first set represents the state of the movement just before the split time begins, while the second set, which is measured while the movement is in green,

Table 2.1. Characteristics of measured variables

Variable	Universe	Measure Unit
number of vehicles	0 - max vehicle	vehicles
occupancy	0- 100	percentage
mean speed	0 - max speed	m/s

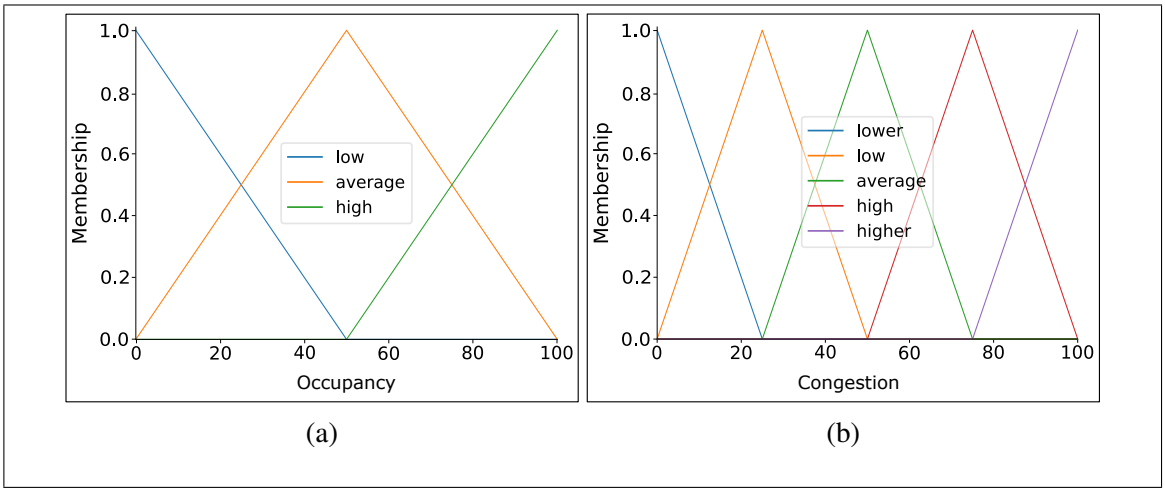


Figure 2.3. a. Fuzzy representation of occupancy; b. Defuzzification of congestion for a given intersection.

relates to how quickly the current state changes. Consequently, the variables *number of vehicles* and *occupancy* belong to the first set, and the *mean speed* belong to the second. The underlying principle is that congestion is a consequence of the relationship between these two groups in the following way:

- If too many vehicles or too much space occupied, then high congestion.
- If too much speed when movement in green, then low congestion.

The opposite holds when the variables go the other way around. Based on this principle, an example rule is:

Algorithm 1 Congestion Model Example Rule

1: **if** $((number\ of\ vehicles$ IS $high)$ OR $(occupancy$ IS $high))$ AND $(mean\ speed$ IS $low)$
2: **then** $congestion$ IS $higher$
3: **end if**

After a figure of the fuzzy representation of the congestion variable is obtained, the crisp value of the congestion state is obtained by applying a centroid defuzzification method over the figure.

Finally, the DTM calculates the congestion for a pair of movements at the supervisor's request, and then communicates the result to it. The full congestion state of the intersection, constructed by the supervisor, is an eight-dimensional vector, where the ith entry represents congestion value of the movement m_i. In case the intersection does not define a given movement, a value of "-1" congestion is assigned, and will be interpreted differently by the neighbors depending if the movement corresponds to an input (value change to zero) or to an output (value changed to the max congestion value). Fig. 2.4 illustrates an example of the state of an intersection. Note that m_4 and m_2 have a motorcycle and a bicycle respectively, but the congestion computed is zero. This is because the occupancy of each of these vehicles is really small, so the output of the congestion model is practically zero for these roads. Nevertheless, these vehicles will still have the opportunity to cross when their movement is activated, even if only for the minimum green time ("$tmin$" in Fig 2.5).

2.3.2.2. TSCM

There are many ways to model an intersection using TPNs depending on aspects such as the controlled variables or extra functionalities. In this work, we take the intersection model presented in List and Cetin (2004) as the base for our developments. This model has been proven to be deadlock free and has properties that facilitate building functionalities on top of it, as: i) its modularity, which allows increasing functionalities by merely adding extra modules; ii) its versatility, allowing to control light changes in a timed and acted way and to configure different sequences of phases inside the cycle of the intersection; and iii) its safety, since it defines phases as pairs of movements that can flow without interrupting each other, then, the TPN only admits safe movement combinations. The TPN is formed

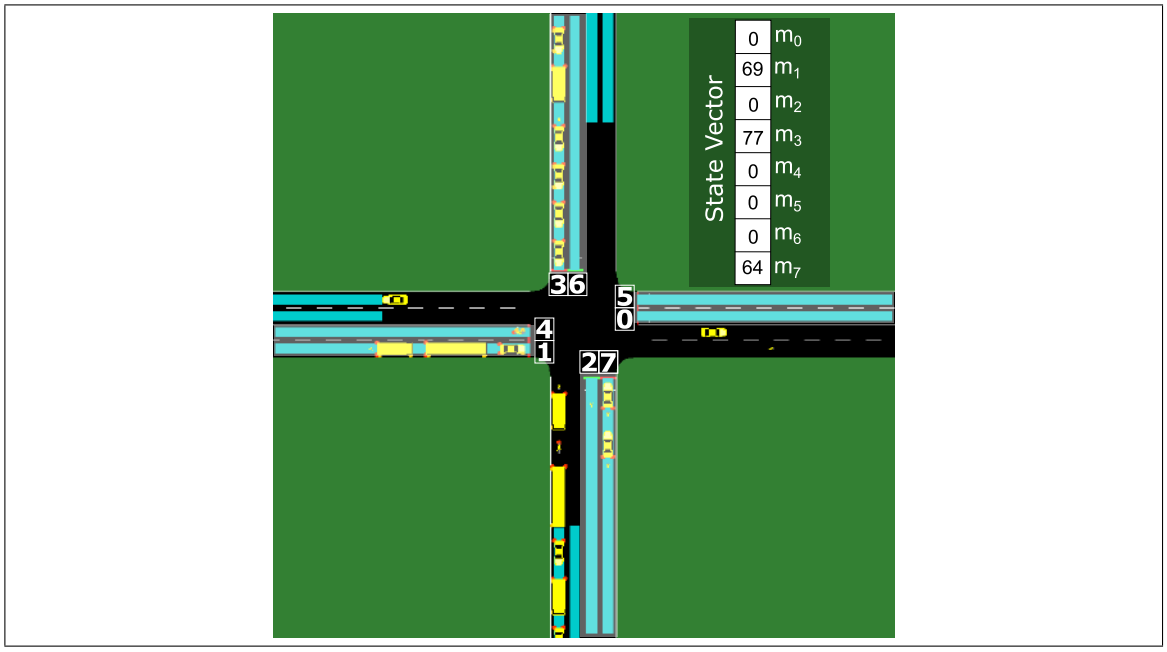


Figure 2.4. Example of the congestion state vector of an intersection that does not have the movement m_1.

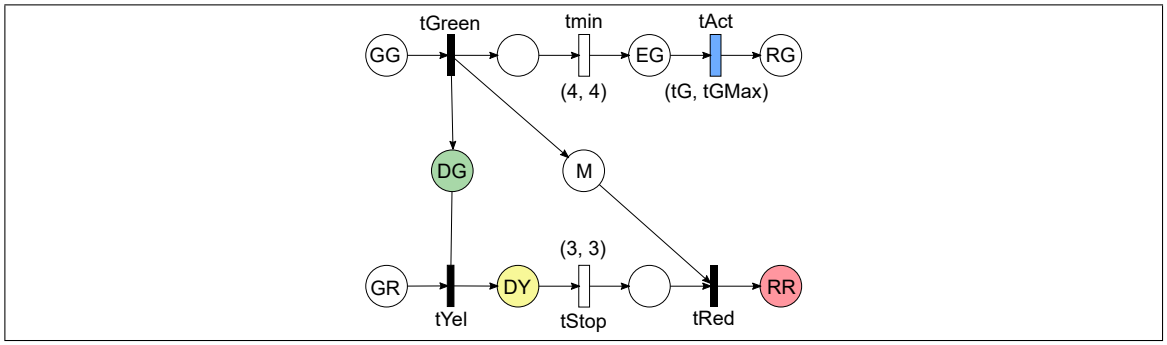


Figure 2.5. Indication Display Module of a movement.

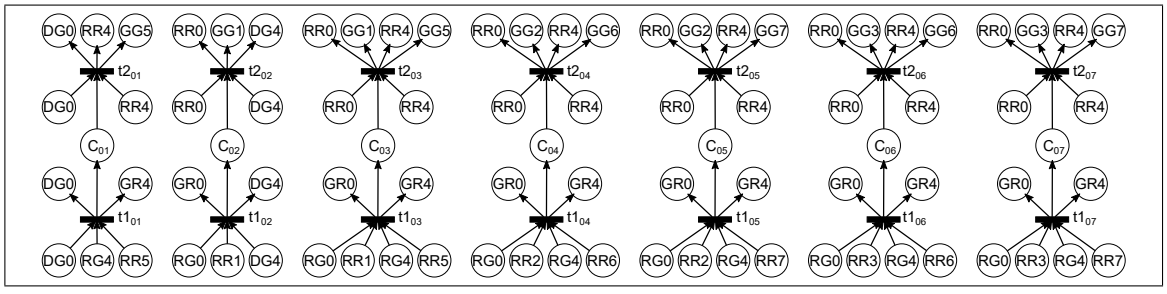


Figure 2.6. PTMs for phase "h_0".

by two module groups: indication display modules (IDMs) and phase-transition modules (PTMs).

An IDM takes care of the logic of the green, yellow and red lights of a given movement m_i. Hence, there must be one IDM, illustrated in Fig. 2.5, per movement. In a nutshell, a movement begins in the *go green* (GG) place; the first transition activates the green light adding a token in place *display green* (DG), and begins a timed process to reach the *rest green* (RG) place, which depends on the EFT and LFT of transitions *min* and *Act*. Then, the duration of the green light of the movement is controlled by the EFT (t_G) parameter of transition *Act*. When the token reaches RG, the PTM removes the token from this place and puts it in place *go red* (GR), which begins a timed process to display the red light (token in place *rest red* (RR)).

The PTMs, on the other hand, are responsible for the transition of the tokens from the RG places to the GR places to initiate a phase transition process. The core of this module is a transition-place-transition set defined as $t1_{xy}$, C_{xy} and $t2_{xy}$ (see Fig. 2.6). There is a PTM per each phase and transition-place-transition set for each possible transition to another phase, giving a total of 56 sets (seven for each PTM) for the intersection shown in Fig. 2.2. Fig. 2.6 shows the PTM of phase h_0, which involves movements m_0 and m_4. There are two different type of arcs connections to the sets, depending on whether the phase change involves the change of a single movement or the change of the two of them. In a nutshell, transition $t1_{xy}$ extracts the token from place RG of movements belonging to the active phase h_x to begin the timed process of changing movements to Red. C_{xy} indicates that the process of changing phase $h_x \rightarrow h_y$ is taking place. Finally, $t2_{xy}$ ends the phase transition process by adding a token in place GG of the movements belonging to the new active phase h_y. A detailed description of this model can be found in List and Cetin (2004).

Note that, as the model only allows the activation of movements in safe pairs (see Fig. 2.2), to control intersections that have vehicle flow in only one movement, this movement must be associated with a non-existent partner in the PN, referred to as “phantom

movement". This phantom movement activates in the PN in pair with the mentioned intersection movement, but is not taken into account for the congestion and split calculation and intersection management.

In this work, we propose to enhance the model in List and Cetin (2004) to allow automatic changes between phases of a cycle. To do so, we propose the addition of a TPN module called "Automatic Cycle Running Module" (ACRM). With this module, the entire TSCM can be implemented in a single device or programming thread, hence increasing robustness and real-time response by not relying on a high-level device for phase changes. Note that this module selects the next active phase according to the module structure. Hence, the decision of which sequence of phases belongs to a cycle has to be taken before system start-up, and be implemented on the design of the "ACRM".

Given a current phase h_x that is ending, a next phase h_y ready to start, and a future phase h_z that follows h_y, the ACRM selects h_z according to the currently active cycle, only when transition $h_x \rightarrow h_y$ begins (i.e., when transition $t1_{xy}$ fires). Fig. 2.7.a shows the four main elements that make up this module. Given a set of different cycles K, the first element of the ACRM module is a set of places, where each place Lk represents an individual cycle l_k ($k \in K$). A Lk place must have a token to activate the transition between phases associated to the cycle l_k. The second is the set of places representing the eight phases admitted by the TPN model, where a place Hj ($j \in \{0, 1, \ldots, 7\}$) contains one token when h_j has been selected as the next active phase within the cycle, i.e., the phase h_y of the next $t1_{xy}$ transition that will fire. The third element are the transitions tLk_{xyz} that perform the token flow process between places Hj, where Lk represents the cycle l_k they are associated to, and x, y, z represent the phases index. This transition must have a small-time (st) (e.g., 1 s) associated, for synchronization with the IDM and PTM processes. The last element is a place called *phase transition condition* (PTC) where a token is placed when any transition $t1_{xy}$ of the PTMs is triggered; its only function is to prevent the next phase selection process from being executed more than once during the current phase changing process.

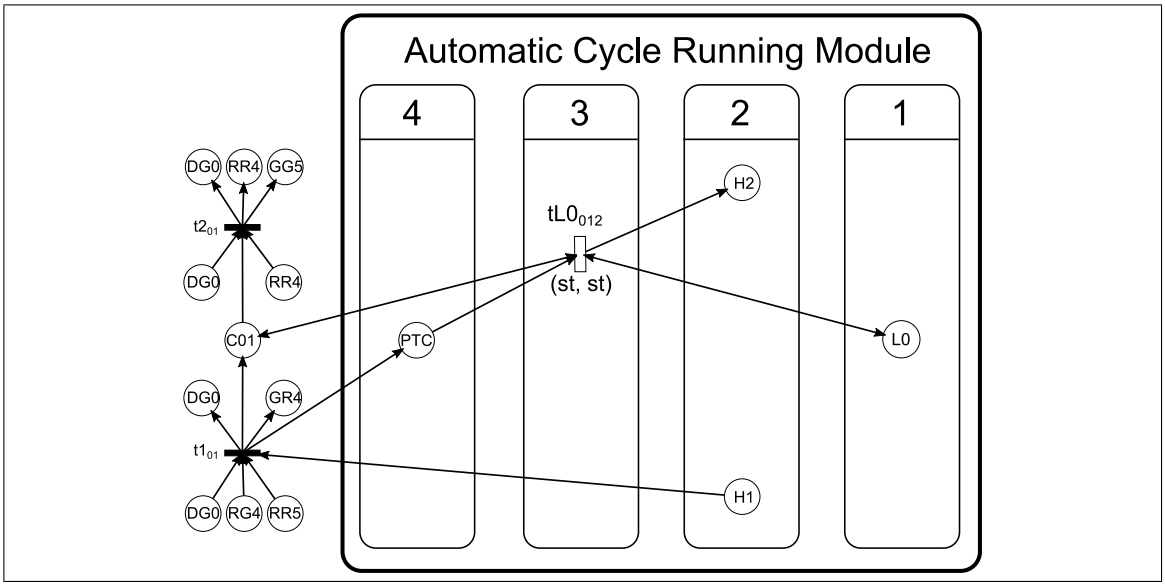


Figure 2.7. ACRM structure and example of transition $h_0 \to h_1 \to h_2$.

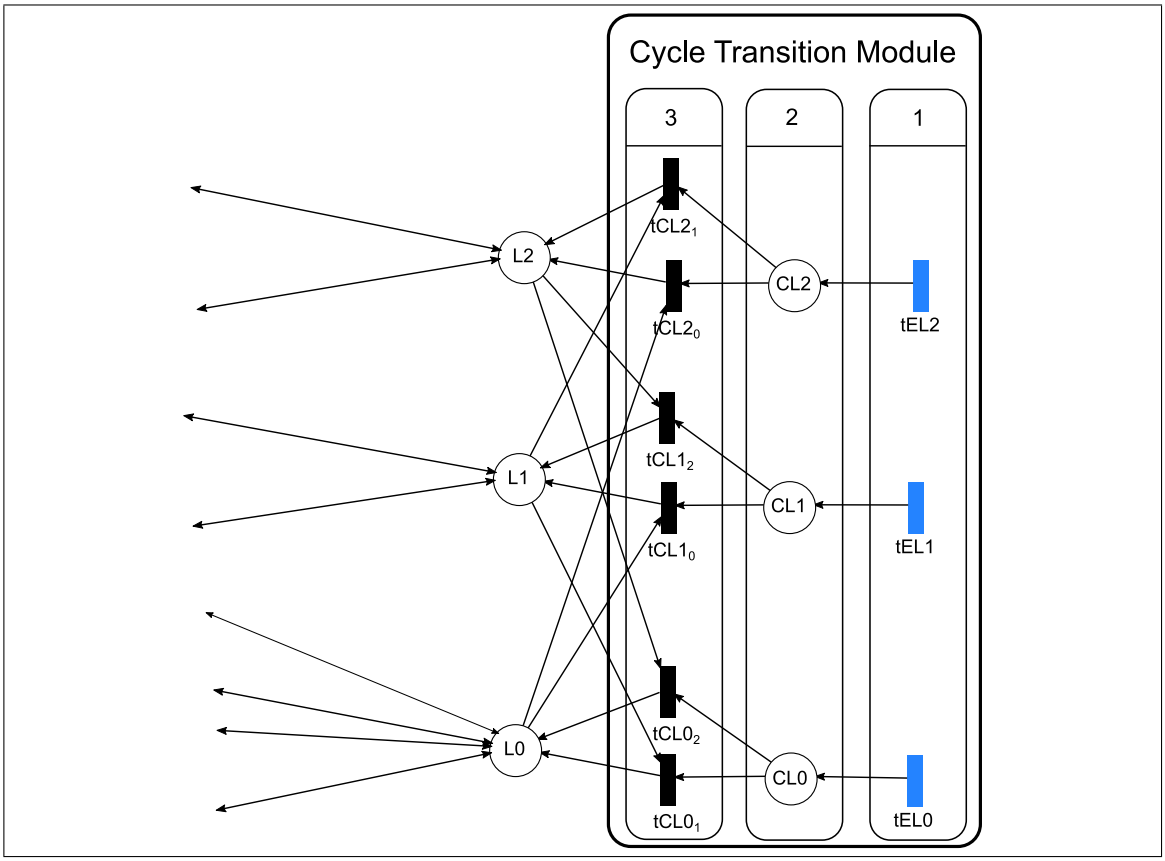


Figure 2.8. Cycle Transition Module structure.

To illustrate the working principle of the ACRM, consider a cycle l_0 with the following phase transitions: $h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow h_4 \rightarrow h_5 \rightarrow h_6 \rightarrow h_7 \rightarrow h_0$. There exists a transition $tL0_{xyz}$ for every allowed phase change, for example, $tL0_{012}$, $tL0_{123}$, $tL0_{234}$, and so on. Regarding connections with other modules: i) the PTC place is connected by an output arc to all the $t1_{xy}$ transitions to indicate the beginning of the transition process between phases h_x and h_y; ii) the Hy places are connected by an input arc to $t1_{xy}$ transitions to establish the firing condition for transitioning to phase h_y; iii) finally, the essence is in the connections of transition tLk_{xyz}:

- Bidirectional connection to place Lk, establishing the firing condition within cycle l_k, without extracting the token from this place so as not to alter the current cycle.
- Bidirectional connection to place C_{xy} at the PTM module, establishing the activation condition of being in the transition process $h_x \rightarrow h_y$, without extracting the token from this place in order not to alter the PTM process.
- Input arc to the PTC place, establishing the activation condition that the phase transition process has been initiated, i.e., $t1_{xy}$ was triggered, and preventing the next phase selection from being executed more than once.
- Output arc to Hz, to assign the new phase h_z as next according to the cycle that is active.

As an example, consider a phase transition from $h_0 \rightarrow h_1 \rightarrow h_2$ in cycle l_0 (see Fig. 2.7). When conditions for switching from phase h_0 to phase h_1 are met, the PTM module takes action activating the $t1_{01}$ transition that i) extracts a token from places $H1$, $RG4$ and $RR5$, and ii) deposits a token in places C_{01}, $GR4$ and PTC (at this moment, the internal transition to the red light of the IDM of movement m_4 starts). This event enables transition $tL1_{012}$, which fires just after the "st" time has elapsed. When $tL1_{012}$ triggers, the token is extracted from place PTC, so that conditions for activating $tL1_{012}$ are no longer fulfilled, and a token is deposited in place $H2$, which is connected to transition $t1_{12}$. With this

last step, the phase-place token condition is reached to carry out the next phase change programmed in the cycle.

The ACRM allows programming multiple cycles, involving different phases, which can be activated upon the occurrence of an event. This scheme allows us to react to a set of catastrophic events or extreme traffic conditions with a cycle change. To this end, an extra module, called "Cycle Transition Module" (CTM), in charge of selecting a new cycle when a event occurs is introduced. Fig. 2.8 shows this module.

The CTM is composed by three elements: i) an *event cycle n* transition $tELk$ that is triggered by an external module (the supervisor in this case) when an event requiring a cycle change, to l_k, occurs; ii) a *change cycle n* place CLk that establishes the change condition from a present cycle to cycle l_k; and iii) a set of *change to cycle n from cycle m* transitions $tCLk_m$ that perform the token flow from Lm to Lk.

To allow the CTM to work in synchronization with the ACRM, we have to add more $tL0_{xyz}$ transition sets that allow the corresponding phase transition from the previous cycle to the new one. Let us illustrate the construction of these sets with an example. Consider two different cycles $L0$ and $L1$, where $L0$ have the phases transitions described earlier in 2.3.2.2, There must be $tL0_{xyz}$ transitions that allow to leave the sequence established by the $L1$ and start $L0$ from its first phase. For example, if $L1$ has the sequence $h_1 \rightarrow h_5$, the part of the module associated to $L0$ has a $tL0_{150}$ transition that will set the conditions to start $L0$ from the beginning (first phase h_0), i.e., it will add a token to $H0$ when phase h_1 is transitioning to h_5. Note that the sequence $h_1 \rightarrow h_5$ will occur anyways because we are selecting the h_z phase h_0 that comes after this transition, but we consider this phase delay is not restrictive for most events needing a cycle transition, as an accident or a change based on the hour of the day.

The dynamics of this module are straightforward since it only seeks to transition the token from one cycle type place to another. Suppose the system is currently running $L0$. Activating the $tEL1$ transition generates a token in the $CL1$ place, which makes the

transition $tCL1_0$ to fire immediately. This action causes a token flow from $L0$ to $L1$, which ends the cycle change. Note that, from the ACRM (see Fig. 2.7), places of type Lk are connected to the phase selection transitions tLk_{xyz} that only belong to phases of this cycle, then the network will be reconfigured at the following phase change to follow the new cycle.

With these two new modules the TPN can be programmed in a micro-controller with high security and robustness standards to control the intersection automatically, safely and in real time, with the ability to receive external commands to modify the splits and also the cycles, upon the occurrence of an event.

2.3.2.3. Supervisor

As explained in Section 2.3.2.2, each movement has an IDM that manages the rotation of its traffic signals. Recall that, when a movement is active, a token flows through the left branch of the module illustrated in Fig. 2.5. Also, recall that after the *min* transition fires, the token is housed in place EG, waiting for the *Act* transition to enable and fire at t_G or when the maximum green time t_{GMax} expires. The extended split time $t_G \in [0, t_a]$ is the manipulated variable the supervisor uses as control action.

Under this premise, it is proposed to implement two intelligent control strategies, denoted P (proportional) and PI (proportional + integral), using congestion information from neighbors and the local congestion, to calculate the next value of the trigger time associated with transition *Act*, here represented as the split extension. Both strategies calculate a delta (positive or negative) value that, in the P strategy, is used to perform a direct calculation of the split extension and, in the PI strategy, is added to the current split extension value, analogous to a PI controller. This process is performed by the supervisor when a PTM phase change process is communicated by the TSCM. At this moment, the supervisor requests the congestion state of the movements involved to the DTM and calculates the delta split, and consequently the final split, when the results arrives. The supervisor then

Table 2.2. Channels affecting movements for a nominal intersection.

Active Movement	Input Movements	Output Movements
0	[2, 3, 5] East	[3, 6] South
1	[1, 6, 7] West	[1, 4] East / [3, 6] South
2	[4, 5, 7] South	[0, 5] West
3	[0, 1, 3] North	[3, 6] South / [0, 5] West
4	[1, 6, 7] West	[2, 7] North
5	[2, 3, 5] East	[0, 5] West / [2, 7] North
6	[0, 1, 3] North	[1, 4] East
7	[4, 5, 7] South	[2, 7] North / [1, 4] East

sends the split values to the TSCM when the corresponding movements are set to green. Algorithm 2 summarizes the steps to implement the control action.

Algorithm 2 Control process

```
1:  if DETECTOR_CHANGE then
2:      if detector.move.display == GREEN then
3:          save_detector_data(move(detector.move), detector.data, "all")
4:      else
5:          save_detector_data(move(detector.move), detector.data, "group_1")
6:      end if
7:  end if
8:  if NEIGHBOR_STATE_CHANGE then
9:      save_neighbor_state(neighbor.state)
10: end if
11: if PHASE_TRANSITION then
12:     for moves IN next_phase do
13:         move.congestion = congestion_measure(move.detectors)
14:         move.split = split_measure(move.congestion, neighbors)
15:     end for
16: end if
```

To calculate the delta split of every movement, a fuzzy expert system approach is taken. As mentioned, this delta is used to set the trigger time of transition *Act* of the IDM to manage the extended split time. Under this scheme, the relative split time of a movement is the time it takes to reach the necessary condition to change its green display status, i.e., the time a token takes to flow from place GG to place RG.

In this case, the fuzzy variables are the current congestion state of the movement, the average of the congestion states of the neighbors' movements that represent an entry flow,

Table 2.3. Variables used in the control process.

Variable	Universe	Measure Unit
my_congestion	0 - 80	Congestion State
in_congestion	0 - 80	Congestion State
out_congestion	0 - 80	Congestion State
Delta Split P	-11 - 11	sec
Delta Split PI	-2 - 2	sec

and the average of the congestion states of the neighbors' movements that represent an obstacle for the outgoing cars. The supervisor uses the information in Table 2.2 to recognize which channels of the neighboring intersections affect a movement, either as an input flow or as an obstacle to the output flow. Table 2.3 shows the universe of discourse and the units of every variable involved. Five fuzzy sets with triangular membership functions were used for both input variables and delta split.

IF/THEN rules were constructed using the intersection of the three input variables considering as design principle that i) the higher my_congestion is, the higher d_split should be; ii) the higher in_congestion is, the higher d_split should be; and iii) the higher out_congestion is, the lower d_split should be. Applying this principle, a total of 125 rules were synthesized, which define the knowledge base for the fuzzy expert system.

The delta split of a movement, after defuzzification, is a float number that will modify the extension split time of a movement every cycle. In case the intersection does not have information of the input flows or output obstacles from its neighbors, a value of 40 will be assigned to this congestion variable. Moreover, if the value of a congestion variable is "-1", representing the absence of the correspond movement in the neighbor topology, a value of zero will be assigned if its an input flow and a value of 80 will be assigned if its an output.

The relative split time of a movement for every cycle will be: i) P strategy: the trigger time of the *min* transition, plus a central split value, plus the delta split P calculated when the phase transition that activates this movement starts; PI strategy: the trigger time of the *min* transition, plus the previous trigger time of the *ACT* transition, plus the delta split

PI calculated when the phase transition that activates this movement starts. In the tests conducted in this chapter, a maximum split time of 26 seconds was established, since it is sufficient to clear an intersection under normal to heavy traffic conditions. A three seconds yellow time and a two seconds all red time were defined based on state of the art works. The trigger time of the *min* transition was defined as four seconds, so adding the yellow and all red times to the phase time, we are left with 9 seconds minimum time for pedestrian crossing. Finally, the central split value was defined as half of the split range length $\in [4, 26]$, equal to 11 seconds.

As a final thought, note that the modularity of the supervisor, together with the fact of being detector, protocol and communication interface agnostic, makes it possible to change or improve this module with the implementation of other strategies, as RL for the split control, or by adding new applications, as emergency vehicle priority actions, without significantly modifying the rest of the system.

2.3.3. Cloudlet Layer

In this work, no particular application was designed for the cloudlet Layer. However, a node is implemented on which applications can be deployed in the future. In the implemented system, all intersection CPOs communicate their congestion state to the cloudlet node periodically, where the information is consolidated in a database to provide a global view of the current and historical state of the neighborhood.

2.4. Implementation Example

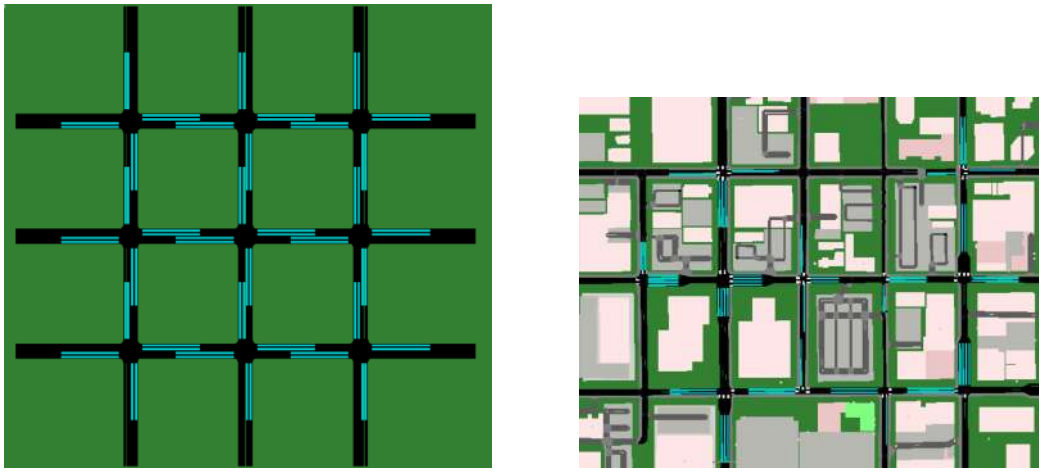
To test the functionality and effectiveness of the strategy, a series of experiments were conducted in a pseudo-real environment, seeking to resemble as much as possible the conditions of the cyber and cloudlet layers to a real application, so switching to a real environment should be transparent.

2.4.1. Experimental setup

Each layer was implemented and tested in the laboratory. The main objective of the experiments is to test the performance of the system in an intersection control application for a neighborhood, emulating conditions as close to reality as possible. To do so, different traffic networks scenarios are simulated in a high-performance computer representing the physical layer; each CPO is implemented in a micro-computer connected to the simulation over a standard wireless interface, from which they receive information from detectors and apply control actions over the simulated traffic signals. The CPOs also connect to each other over the same private network to communicate their congestion state. Finally, a first release of the cloudlet was implemented in a second high-performance computer connected to the CPOs over the Internet. A detailed description layer by layer is given below.

2.4.1.1. Physical Layer

It was decided to simulate the physical layer due to the complexity that exists in implementing the solution in real transportation infrastructure. To this end, a popular simulator called Simulation of Urban Mobility (SUMO) was used. SUMO is an open-source, microscopic, multi-modal traffic simulator, that allows simulating how a given traffic demand, which consists of a set of vehicles, moves through a given road network (Krajzewicz et al., 2012). SUMO has several tools to design traffic networks in a simple way, download real street maps to perform tests on them, generate flows of different vehicle types, implement road detectors and traffic signals to receive traffic information and perform control actions based on this information, generate accidents, among others. One of SUMO's main tools used in tests with control actions is the Traffic Control Interface (TRACI). This tool is a Python package that allows accessing simulation information and applying control actions to its elements (e.g., traffic signals).



(a) (b)

Figure 2.9. a. Ideal scenario: nine intersections fully instrumented; b. Real scenario: section of the city of San Diego with 11 fully instrumented intersections and five not controlled intersections.

Two networks were designed: i) an "ideal network" of nine intersections of equal morphology, with the possibility of managing all the movements established in this chapter (see Fig. 2.9a). Each lane has a max speed of 13.89 m/sec (50 km/hour) and a length of 80 meters with a lane area detector covering a 62.5% of it (50 meters) that gathers the three variables needed to calculate the congestion model. In this scenario, each node has at least two instrumented neighbors with which it can exchange information; and ii) a realistic scenario extracted from the "OpenStreetMap" database (OpenStreetMap contributors, 2017), which represents a section of the street map of the city of San Diego, California. This scenario has 16 intersections, of which 11 are controlled (see Fig. 2.9b) and the other five have no traffic signals. The max speed of all lanes is 13.89 m/sec (50 km/hour). It also has some particular characteristics such as intersections with different morphologies, lanes with different length, intersections that are not instrumented and intersections with a single neighbor to share information with. These characteristics make it a complicated scenario, hence ideal to test the performance of the strategy in unfavorable conditions.

Table 2.4. Vehicles inserted during the simulation length of half an hour for each vehicular flow setup

	Free	Low	Moderate	High
Ideal	707	804	880	1047
San Diego	587	947	1145	1292

For both scenarios, four different setups were tested: free, low, moderate and high congestion. All setups are configured to insert vehicles in the simulation with predefined routes for half an hour (see table 2.4); however, this time can be extended if due to traffic reasons the simulation cannot place a vehicle at the beginning of its route at the established time. In this case, the simulation will wait for the route to clear before placing the vehicle. Both setups include five types of vehicles: cars, motorcycles, bicycles, trucks and buses. Each type of vehicle has its own configuration regarding size and maximum speed. For detailed information the reader is referred to Guzmán (2019).

All the scenarios were run on a high end computer with a Intel(R) Core(TM) i7-7700 Quad-core (8M Cache, up to 4.20 GHz) processor and 32GB DDR4 RAM.

2.4.1.2. Cyber Layer

Each CPO is composed by three different Python scripts, one per each module, in line with the modular architecture proposed earlier. These modules interact with each other and with the simulation to manage the traffic.

Five split management strategies were tested: the two intelligent strategies previously explained, an optimal control strategy based on the Webster method (Zhihui et al., 2018), a coordinated strategy based on traffic signal offset calculation to create green waves, and a baseline timed strategy where all the splits are equal to the average of the minimum and maximum limits of the intelligent strategies (15 seconds). The intelligent strategies were implemented using the Scikit-Fuzzy (Warner & Sexauer, 2019) python library, while the others use tools provided by the SUMO software.

Table 2.5. Technical Specifications of devices used at the cyber layer

	RPi 3B	RPi 3B+	BBG	BBBW
Processor	Broadcom BCM2837 64bit CPU, Quad Core 1.2GHz	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz	AM335x 1GHz ARM® Cortex-A8	Octavo Systems OSD3358 1GHz ARM® Cortex-A8
RAM	1GB LPDDR2 SDRAM	1GB LPDDR2 SDRAM	512MB DDR3 RAM	512MB DDR3 RAM
Interfaces	2.4GHz 802.11b/g/n	2.4GHz and 5GHz IEEE 802.11.b/g/n/ac	2.4 GHz 802.11b/g/n	2.4 GHz 802.11b/g/n
Storage	32GB SanDisk Ultra micro-SD CLass-10	32GB SanDisk Ultra micro-SD CLass-10	4GB 8-bit eMMC on-board flash storage	4GB 8-bit eMMC on-board flash storage

Each CPO was implemented in a micro-computer with moderate processing capabilities that can connect to two or more interfaces at the same time. Four different models of devices were used, running the same application, to verify that the strategy does not depend on the hardware. These devices are, Raspberry Pi 3B (RPi 3B), Raspberry Pi 3B+ (RPi 3B+), BeagleBone Green (BBG) and BeagleBone Black Wireless (BBBW). Table 2.5 shows the technical specifications of the devices. For the ideal scenario we use two RPis 3B, three RPis 3B+, one BBG and three BBBWs, and in the real setup a total of two RPis 3B, three RPis 3B+, two BBGs and four BBBWs.

The TSCM TPN was implemented using the PNs python library SNAKES (Pommereau, 2015), which provides all the required tools to design many types of PNs and run them in real time.

There is a configuration file shared by the three modules that contains all the parameters necessary to configure the CPO to an specific intersection morphology, and it has to be set for every intersection before its implementation. This file includes the intersection ID, the movements it manages, the number of lanes of each movement, the phases for each configured cycle, the neighbors relative position and IDs, the detectors and traffic signals IDs, among others. For detailed information and source code the reader is referred to Guzmán (2019)

2.4.1.3. Communications

The CPO modules interact internally using an open-source universal messaging library called ZeroMQ. A publish-subscribe messaging pattern was used due to its performance distributing time-sensitive information efficiently, to its versatility, and to its widespread use in applications related to cooperating objects. (Rodríguez et al., 2016)

Communications between the simulation and the cyber intersections take place over a WiFi network using the MQTT protocol. All detectors of an intersection are grouped into a single topic by the simulation and the nodes subscribe to the topic of their own detectors. The detector is then identified by the message ID. Every time a value changes in a detector, the simulation publishes a json with the current detector state to the corresponding topic using a data model based on the *Fiware-Datamodels* for transportation data, as follows.

```
Fiware-Datamodel {
    id: Detector ID
    type: TrafficFlowObserved
    laneId: Lane of the detector
    location: Location of the detector in format
            GeoProperty.geo:json
    dateObserved: Date of observation
    occupancy: Occupancy of the detector area
    meanSpeed: Mean speed of vehicles in the detector
    vehicleNumber: Number of vehicles in the detector
    laneDirection: Direction of the lane}
```

To communicate the cyber layer and the cloudlet layer, a similar strategy was used thanks to a DeviceHive plugin (see next paragraph for more information regarding DeviceHive usage) that handles the authentication of the nodes and stores their information using MQTT messages. Each CPO subscribes to the DeviceHive node with a token generated by the platform. Then, each CPO publishes a json with its state through MQTT using

the same ID with which it was registered and the DeviceHive platform is responsible for storing the info of each node in a database for later use.

The communication with the simulation and between the intersections runs over a private WiFi network managed by a high end router (Tp-Link Archer c5400x) to ensure a fast information flow and a good QoS. CPOs connect to the Internet through a different private WiFi network managed by a Linksys WRT1900ACS router, to communicate with the cloudlet layer located in a different subnet.

2.4.1.4. Cloudlet Layer

The cloudlet is implemented in DeviceHive, an open source IoT platform whose main features are:

(i) allows managing large networks,
(ii) provides authentication with JWT (Jason Web Tokens),
(iii) communication is done using web sockets, with the use of an API REST, by MQTT through an API MQTT,
(iv) has a database for storing metadata,
(v) has a connector for an external Cassandra database,
(vi) provides an API in different programming languages for integration of micro-services.

The Cassandra connector is used to store the data flowing from the intersections. Cassandra is a highly scalable and flexible database, which can be easily accessed by several open source applications for big data analysis and machine learning applications. In this way, the basis for implementing the higher level functionalities described in Section 2.2.3 are achieved.

The cloudlet was deployed on a high-performance computer with the following specifications: Intel(R) Core(TM) i7-7700 Quad-core (8M Cache, 4.20 GHz) processor, 32GB

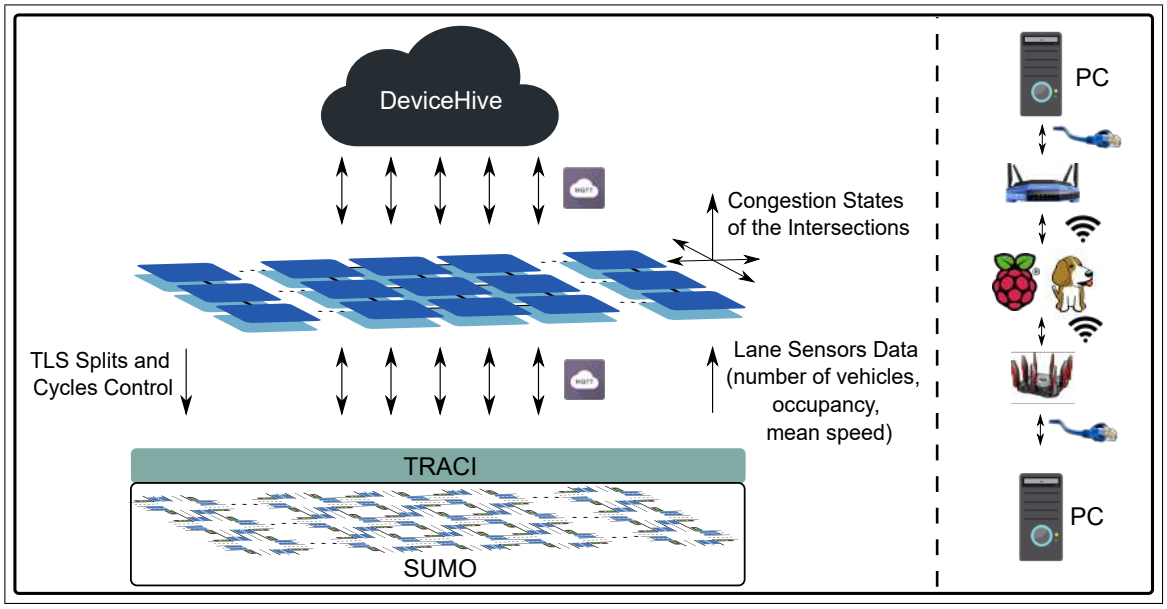


Figure 2.10. Experimental setup implemented to evaluate the proposed intersection management strategy based on a CPS approach.

DDR4 RAM and NVIDIA QUADRO P400 2GB DDR5 graphics. Fig. 2.10 illustrates the experimental setup built for this study.

2.4.2. Experimental Results

To evaluate performance of the intersection management system, the following indicators were used.

- Speed: Average vehicle speed in m/s
- Trip Duration: Average vehicle trip duration in seconds
- Wait Time: Average time a vehicle has a speed of less than 0.1 m/s
- Time Loss: Average time lost due to driving below the ideal speed (slowdowns due to intersections, etc.)
- Depart Delay: Average delay time for vehicle insertion into the simulation

Fig. 2.11 presents the results obtained for the five evaluated control strategies over the four "Ideal network" congestion scenarios. In this network, with the configuration

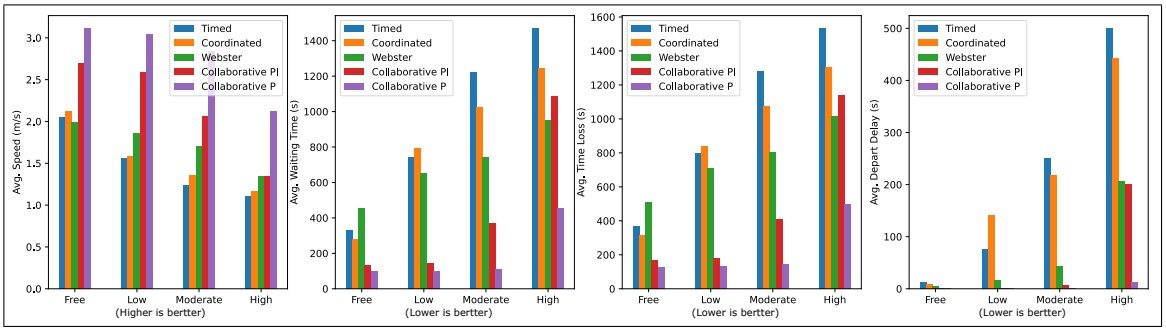


Figure 2.11. Average of the experimental results of the five control strategies tested in the four "Ideal Network" congestion scenarios.

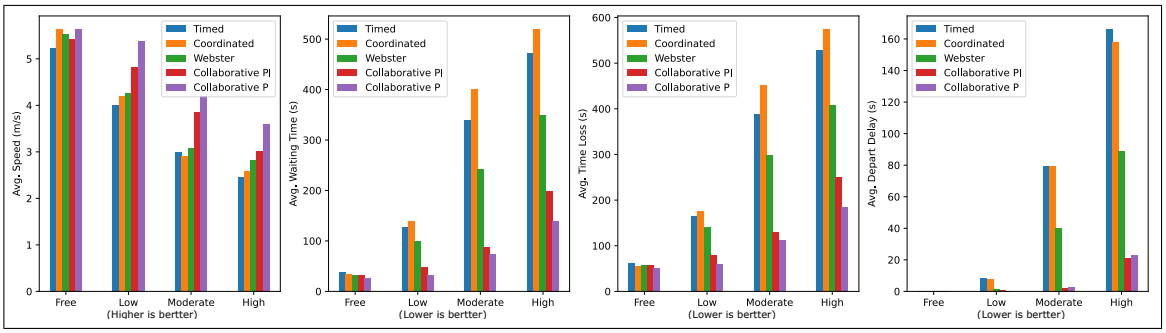


Figure 2.12. Average of the experimental results of the five control strategies tested in the four "San Diego Network" congestion scenarios.

shown in Fig. 2.9a, it can be seen that the proportional intelligent controller outperforms the rest of the strategies for all the indicators in all the setups. The PI approach have a good behavior in the first three scenarios, but in the last one, it gets beaten by the Webster method in the first three indicators. This is because both intelligent strategies deteriorate exponentially against the introduced vehicle flow. However, this exponential deterioration has a slow rate and all the other strategies get the roads saturated before the intelligent approaches, so there is a long gap of vehicular flow where the proposed solutions are significantly superior than the others, particularly the proportional strategy.

It has to be highlighted that the collaborative proportional strategy performs 93% better than the others in the "depart delay" indicator, which implies a more efficient use of the network and fewer time for the simulation to end. This can be also observed in Fig. 2.13,

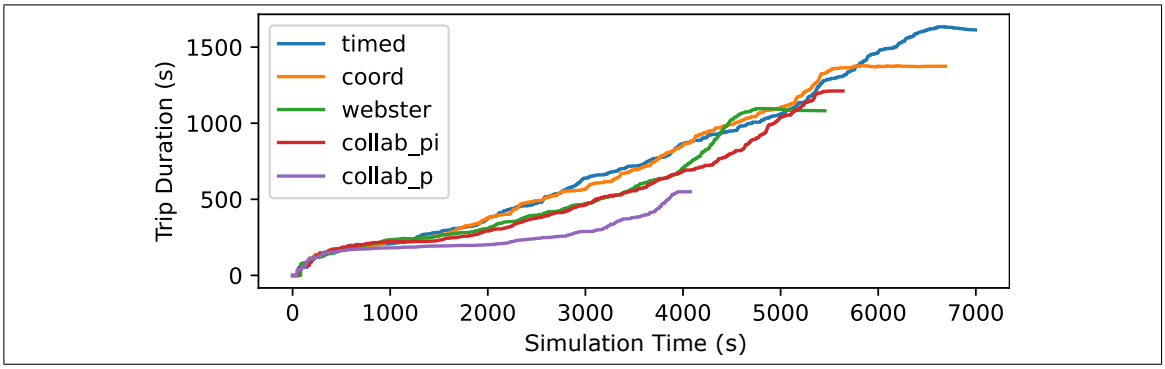


Figure 2.13. Vehicles Mean Trip Duration Time as a function of the simulation time (lower is better) in the "Ideal Network" high congested scenario

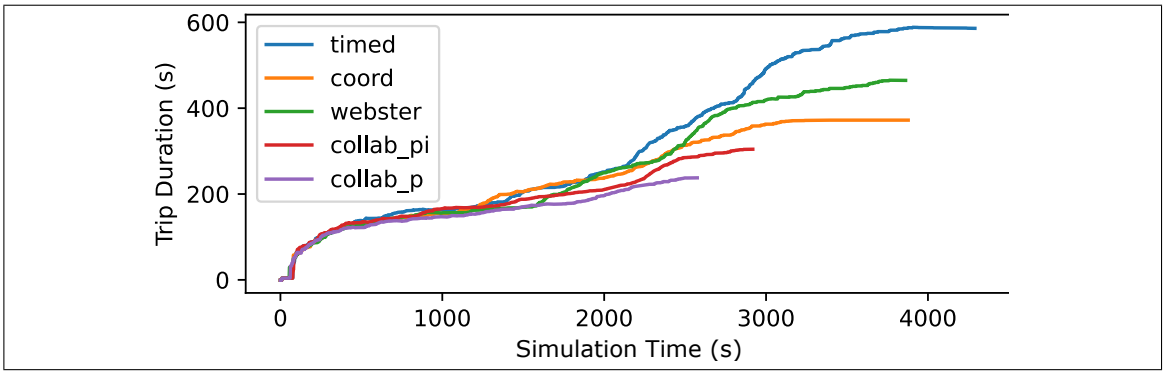


Figure 2.14. Vehicles Mean Trip Duration Time as a function of the simulation time (lower is better) in the "San Diego Network" high congested scenario

where the mean of the trip duration time of all the introduced vehicles is plotted as a function of the simulation time for the five controlled strategies in the high congested scenario. It can be seen that all the curves start at the same point, but they separate as the simulation progresses. Naturally, methods that achieve shorter mean trip duration times end the simulation much sooner than the others, which results in curves having different lengths in Fig. 2.13; this is particularly notorious for the collaborative proportional method, which ends more than twenty two minutes (25% time less) before the next strategy.

A similar behavior is observed in the results of the "San Diego Network" tests shown in Fig. 2.12. First of all, the collaborative strategies still outperform the others in all the congestion setups. However, in this case both strategies present a more linear behavior

for the test indicators against the vehicular flow, with a lower slope than the other methods, which validates its possible use in many congestion setups over real world scenarios. Second, it is interesting to notice that the coordinated strategy has the worst performance of all the tested strategies in almost all the scenarios. This can be attributed in part to the fact that there are five intersections without traffic lights, which makes it really hard to perform a coordinated wave of vehicles. That, added to the fact that there may be no significant vehicle flow in some specific directions to generate substantial green waves, makes this approach inefficient in many urban scenarios. Finally, in Fig. 2.14, both collaborative controllers have a slower increasing behavior compared with the rest and end twenty (collab_p) and eighteen minutes (collab_pi) before the next controller.

It has to be noted that the proportional strategy achieves much better results than the PI. This may indicate that real time data and timely actuation is more important than actions taken with previous data, at least for collaborative control. For this reason, the proportional approach was used for extra tests.

To evaluate the robustness of the solution, a test over the collaborative proportional method was made, where some of the CPOs where disabled and replaced with a regular timed controller, with the split equal to the mean between the min and max split times (15 seconds). This test represents an scenario were the "supervisor" of some of the intersections stop working, loosing the communication with these nodes and keeping the traffic signal running with a fixed cycle using only the more robust timed petri net based TSCM. For the ideal network, nodes two (North-East) and six (South-West) where disabled (22% of the nodes), and for the San Diego Network, nodes two (West), four (Center) and ten (South-East) were disabled (27% of the nodes). Table 2.6 shows the performance comparison between this strategy and the strategy with all the nodes operating. It is also compared with the Webster method that achieves the best performance between the regular strategies. In all the cases, it is observed that the collaborative strategy with disabled nodes has a slightly worse performance than the complete solution, but it still outperforms the Webster method. As expected, the difference between the partial solution and the complete one

Table 2.6. Experimental results of the webster method, the collaborative proportional (col_p) strategy and the collaborative proportional with disabled nodes (col_p_nd) strategy for both Ideal and San Diego networks under decongested and congested traffic patterns.

			Speed ↑	Waiting time ↓	Time Loss ↓	Trip Duration ↓
Ideal Network	Low	webster	1.86	650.06	706.72	763.26
		col_p_nd	2.88	114.21	147.21	194.93
		col_p	**3.04**	**100.34**	**131.81**	**179.23**
	High	webster	1.34	950.17	1017.7	1081.86
		col_p_nd	1.92	638.59	683.16	744.42
		col_p	**2.12**	**454.73**	**494.65**	**552.79**
San Diego Network	Low	webster	4.27	100.08	139.86	191.98
		col_p_nd	5.13	39.45	66.41	118.67
		col_p	**5.38**	**33.14**	**59.59**	**111.87**
	High	webster	2.82	349.82	408.78	464.77
		col_p_nd	2.9	285.21	337.2	392.15
		col_p	**3.6**	**138.75**	**184.47**	**237.72**

gets larger as the vehicle flow grows (worst indicator is the wait time with 28% difference in the Ideal Network and 51% in the San Diego Network), yet it still is better than the Webster in all the evaluated setups. This test validates a robust performance when at least 70% of the system is working, which is closer to a real scenario where some intersections may not be instrumented or there may be problems with the CPOs.

2.5. Discussion

In this chapter a three layer CPS approach to CIM is proposed, with a modular cyber layer that facilitates the design of systems using this architecture and provides the required functionalities for its implementation in a real world scenario, such as: i) a DTM that receive the traffic data from the physical layer to model a "congestion" variable representing the state of the intersection, making the rest of the system detector agnostic; ii) a Timed Petri Net based TSCM that acts as a robust and safe traffic signal cyber actuator, and allows implementing external strategies for CIM and event reaction; and iii) a Supervisor module in charge of implementing intelligent control strategies and communication with neighboring intersections and high level applications. To validate the architecture,

from a functional perspective, an instance was designed and implemented, using Expert-Fuzzy strategies for congestion modeling and split calculation. Evaluation in two scenarios shows that the proposed approach can efficiently handle CIM, outperforming timed, webster and coordinated strategies.

Despite the promising results, several limitations exist, which are discussed in the following.

- In section 2.4.2 it was mentioned that the collaborative strategy with disabled nodes can still outperform the Webster method, when at least 70% of the intersections are controlled with the proposed P strategy. In a real implementation, there may be situations where the number of intersections that can be controlled is less than this percentage, either due to the lack of instrumentation or equipment failures. For this reason, it is recommended to carry out sensitivity studies regarding the disconnection of nodes in different scenarios, in order to find the limit where it is no longer convenient to use the proposed solution.
- A point that was not developed in detail in this research was the different data and information sources that can be used as input for the control strategy. In this work, it was decided to use the traffic variables of speed, occupancy and flow, which can be measured with conventional traffic detectors. However, there are many other interesting options to take into consideration, such as: predictive traffic information, information from cars with data transmission units, information from autonomous vehicles, pedestrian traffic information, among others. As mentioned in section 2.2.2.1, one of the advantages offered by the implementation of the DTM module is that an intersection state variable is generated from the physical layer data, which makes the rest of the system agnostic to the types of data and detectors used. Therefore, the most straightforward method for incorporating new data sources is to design a model for generating the intersection state variable using this data as input, without making significant changes to the rest of the solution.

- Pedestrians flow is a particular case of data that is interesting to discuss on its own, because in some occasions pedestrian flow at an intersection can be more abundant and significant than vehicular flow. Although, as mentioned in 2.3.2.3, the design of the TSCM module establishes a 9 second "active" time per phase p_i so that pedestrians associated with the other phases can transit, it would be interesting to take the pedestrian flow data as one of the input variables to define the split of the traffic signals. As mentioned in the previous point, the most direct option would be to incorporate this data as input to the DTM model. Another interesting option is to increase the dimensions of the state variable output by the DTM, to generate a vector related to the vehicular flow and another related to the pedestrian flow associated to each phase. This increase in dimensions would generate a richer input for the CIM controller, but it would create a constraint that all intersections must be properly instrumented to detect pedestrian flow, so that this new state vector can be generated and shared with neighbors to take control actions.
- One difficulty for implementation of this autonomous systems in real environments is the fact that drivers are used to a standard cycle duration and to a distribution of phases being more or less constant over time. For this reason, it would be interesting to discuss some ways to facilitate user acceptance. As an initial solution, it is proposed to implement signs that allow the user to identify the autonomous feature of the traffic light (e.g. a sign on the traffic light that says "Intelligent Traffic Light" or "Autonomous Traffic Light"). In this way it will be more difficult for the user to confuse the behavior of the traffic light with a malfunction. On the other hand, with respect to the system's ability to react to events, it would be a good option to implement screens that allow users to be informed of the occurrence of such events (e.g., road status screen), so that users are aware of the current situation and can take appropriate action.

In closing, it is important to state that a natural steps for extending the work presented in this chapter include the design and implementation of a high-level application to test the

capabilities of the cloudlet layer of the proposed architecture, and to improve the CIM control strategy by designing and implementing a more sophisticated data-driven controller, that uses, for example, learning-based algorithms to define the splits. These two directions will be developed in the next two chapters of this thesis.

3. REAL-TIME TRAFFIC PREDICTION IN LAS VEGAS I-15 FREEWAY

This Chapter details the development of traffic prediction application that makes use of the architectural paradigm introduced in the previous chapter. Therefore, it contributes to the achievement of specific objective (iii). A highlight is the implementation over a real environment in the city of Las Vegas[3].

3.1. Context

Within an ITS, traffic prediction plays a key role for traffic management, since it allows adjusting routes and systematically allocating resources (Ma et al., 2017). However, traffic prediction is a rather complex task due to the difficulties in modeling traffic patterns in urban areas and the numerous unmodeled disturbances that take place during the day. Recently, thanks to the exponential increase in the number of detectors implemented in the transportation infrastructure, addressing the traffic prediction problem using data-driven techniques has emerged as a viable option. Data-driven techniques allow modeling urban traffic in an accurate way, and thus enable obtaining accurate short-term predictions useful for planning (Li et al., 2018; Lana et al., 2018).

Among the new data-driven techniques used for traffic prediction, deep learning (DL) models have obtained the best results, thanks to their ability to find complex non-linear relationships between data points, which are very difficult to extract with simpler models (Yin et al., 2021). Specifically, models that work with spatio-temporal data, such as Convolutional Neural Networks (CNN) and Graph Convolutional Neural Networks (GCNN), excel when predicting traffic (Ma et al., 2017).

Concretely, in Wang et al. (2016), a novel architecture called “Error-feedback Recurrent Convolutional Neural Network” (eRCNN) is presented, which uses information of

[3]The material in this chapter is published in “Guzmán, J., Morris, B., & Núñez, F. (2022). A cyber-physical system for data-driven real-time traffic prediction in Las Vegas I-15 freeway. IEEE Intelligent Transportation Systems Magazine. doi: 10.1109/MITS.2022.3211996” (Guzmán, Morris, & Núñez, 2022).

the prediction error from previous time steps to improve subsequent predictions, via the error-feedback recurrence method. The proposed eRCNN uses as input a spatio-temporal matrix of vehicle speed measurements and errors from previous predictions to predict the vehicle speed (used as traffic level indicator) at a given detector at a given time. Two datasets were created in Wang et al. (2016) using vehicle speed measurements, sampled with a 5 minutes sampling period, from detectors in two Beijing roads: the 2nd ring road and the 3rd ring road. Data was collected during 25 weekdays in November 2013, yielding approximately 7,200 samples. The first dataset considers measurements from 88 road segments, while the second includes measurements from 122 road segments. A comparative analysis conducted in Wang et al. (2016) showed that the eRCNN outperforms ARIMA, Support Vector Regression (SVR) and CNN models in predicting vehicle speed.

In Mena-Oreja and Gozalvez (2020), a comparative study was performed between several state-of-the-art DNN models for traffic prediction based on spatio-temporal images. The evaluated architectures were setup to receive as input spatio-temporal measurements of flow, occupancy and vehicle speed, and included models using previous prediction errors via the error-feedback recurrence method, the eRCNN proposed in Wang et al. (2016) among them. The datasets used for training and evaluation consisted in two years of data, 2015 and 2016, from three different segments of the I5 freeway in California: I5-N-3 (27 detectors), I5-S-3 (25 detectors), and I5-S-4 (31 detectors). Data was collected with a 5 minutes sampling period, yielding a total of 209,998 samples for each dataset. Results presented in Mena-Oreja and Gozalvez (2020) show that models using the error-feedback recurrence method outperformed their competitors, particularly in congested conditions, the eRCNN being the best performer of the study.

In Li et al. (2018), a novel GCNN architecture called “Diffusion Convolutional Recurrent Neural Network (DCRNN)” is presented. The DCRNN captures spatial dependency using bidirectional random walks on the graph, and temporal dependency using an encoder-decoder structure with scheduled sampling. Two datasets were used for training and evaluation. The METR-LA dataset consists in vehicle speed data of 207 detectors

in Los Angeles County, taken every 5 minutes from March 1, 2012 to June 30, 2012. Similarly, the PEMS-BAY dataset consists in vehicle speed data of 325 detectors in the California Bay Area, taken every 5 minutes from January 1, 2017 to May 31, 2017. Comparative results presented in Li et al. (2018) show that the DCRNN outperforms ARIMA, SVR, and Recurrent Neural Networks with fully connected Long short-term memory (FC-LSTM) models in predicting vehicle speed for prediction horizons ranging from 15 to 60 minutes.

In Wu et al. (2019), a GCNN model called Graph WaveNet (GWNet) is presented, which was designed based on the Wavenet model for raw audio generation. The GWNet uses stacked blocks of dilated 1D convolutions and GCNNs to capture the hidden temporal and spatial dependency in the data. For training and testing, the METR-LA and PEMS-BAY datasets from Li et al. (2018) were used. Results show that the GWNet outperforms the DCRNN proposed in Li et al. (2018) for different prediction horizons ranging from 15 to 60 minutes.

Finally, in Yin et al. (2021), a comparative study of state-of-the-art data-driven models, including a new variety of GCNN models, is presented. For training and testing, the METR-LA and PEMS-BAY datasets were used. Results show that the GWNet model outperforms its competitors in predicting vehicle speed for prediction horizons of 15, 30, and 60 minutes.

Despite the good results of DL models documented in the literature, an important issue is that most studies have been conducted in controlled scenarios, with validated datasets that have undergone preprocessing and cleaning stages. A concrete implementation capable of predicting traffic in real time, dealing with phenomena such as missing data, data corruption, sensor failures, imperfections in communication networks, and sudden traffic pattern changes is currently not available in the technical literature.

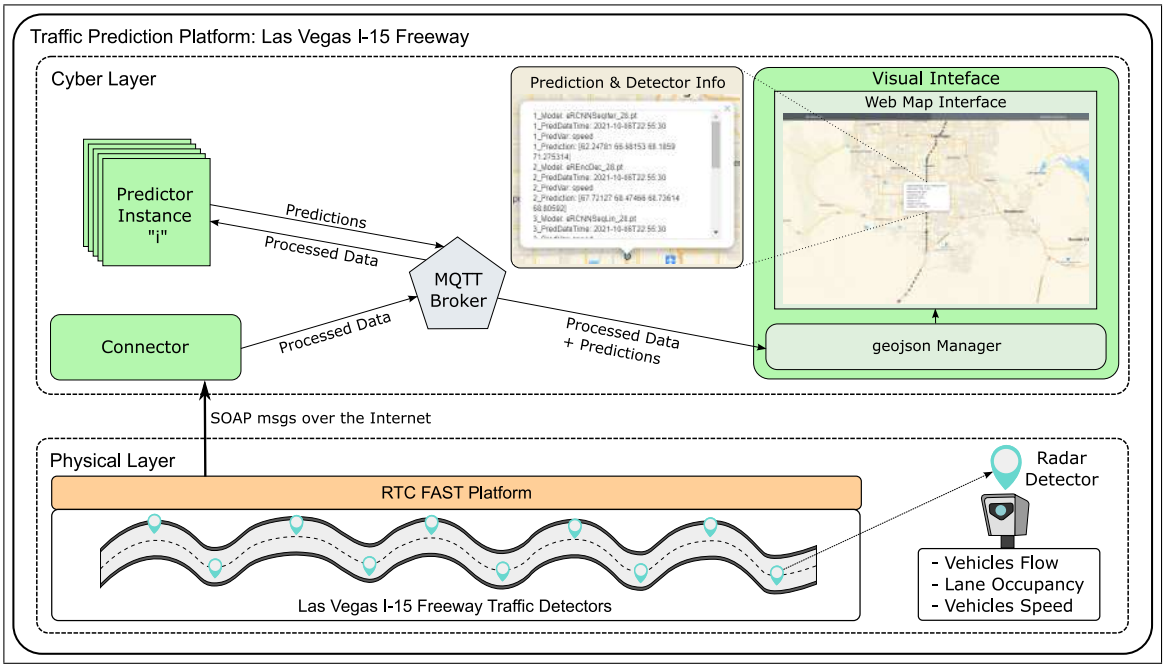


Figure 3.1. Schematic of the Real Time Traffic Forecasting Platform Architecture. The green boxes represent the platform service, the orange box represents the external data source and the grey diamond represents the broker for the internal communication.

3.2. CPS Architecture

The architecture used follows the general paradigm introduced in the previous chapter consisting in three layers: i) a physical or field layer, where data is generated; ii) a connectivity layer; and iii) a cyber layer, where data is interpreted, processed and visualized. Fig. 3.1 presents an overview schematic of the CPS; each layer is discussed in the following.

3.2.1. Physical layer: FAST

The FAST platform from the Regional Transportation Commission of Southern Nevada (RTC) is a traffic monitoring and control system that manages and integrates a series of detectors deployed in the city of Las Vegas. FAST exposes, via standardized communication interfaces, the data from a variety of traffic detectors, mostly radar and video cameras, deployed along four of the main freeways in Las Vegas: I-15, US-95, I-215, and CC-215.

Detectors in FAST generate measurements of most of the classical variables used in data-driven traffic prediction, namely, vehicle volume, occupancy, and vehicle speed. Hence, all of the state-of-the-art prediction models can be implemented using data from FAST.

In this work, we focus on data-driven traffic prediction using spatio-temporal images created from traffic data, which is the state-of-the-art method for traffic prediction as mentioned in Section 3.1. A spatio-temporal image is a representation of the data where each variable represents a channel of an image, the value of the variable of interest is represented as a pixel intensity and the 2D position of the pixel is interpreted as the spatial (position of the detector generating the sample with respect to the other detectors in the system) and temporal (with respect to the time span represented in the image) information of the sample.

Using spatio-temporal images is beneficial for exploiting spatial and temporal relationships in the data; yet, a limitation is that the models can only work in (practically) linear segments of highways, since their implementation for intersecting roads is not efficient. For this reason, in this work we focus on a single segment of the I-15 freeway, one of the most important and busiest roads in Las Vegas.

3.2.2. Connectivity layer: SOAP

To interact with FAST, the Simple Object Access Protocol (SOAP) is used. SOAP is a request-response protocol that provides the Messaging Protocol layer by defining commands for a client to access the services exposed by a second node. SOAP uses XML Information Set for its message format and relies on application layer protocols for transport. FAST exposes measurements from detectors through SOAP at a sampling rate of one minute.

To have more control over the communication management, it is proposed to implement a local service that will be in charge of data handling and processing. This service will be explained in detail in the next subsection.

3.2.3. Cyber layer: Connector, visualization, and predictor instances

The cyber layer implements a series of services for handling and processing the data. This facilitates the design of applications based on this architecture, since each service can be designed independently from each other, as long as the data structure of the communication is consistent. Consequently, the cyber layer can be broken down into three main parts, which are described in the following.

3.2.3.1. Connector

The connector is the service in charge of managing the data ingestion from the FAST platform. The connector serves as a gateway bridging the SOAP interface to an MQTT-capable interface that exposes the data to be consumed by the other services in the cyber layer in the form of a json file. MQTT is a publish-subscribe protocol with low overhead that reduces the communication load inside the cyber layer, compared to SOAP that typically uses HTTP for transport. In addition to serving as a protocol translator, the connector performs simple data-transformation actions to prepare the data for further processing. Specifically, the connector:

- Aggregates the data coming from the different lanes of the freeway, generating only one value for each variable per detector.
- Discards the information coming from detectors that are not used by services in the cyber layer, i.e., generates a subset of detectors to be used for prediction.
- Translates the “Speed” variable from miles per hour to kilometers per hour (this is necessary since the historical dataset used for training uses kilometers).

- Re-samples the raw data coming at one sample per minute to generate a data stream sampled at one measurement every 15 minutes. This is desirable since historical data is consolidated in FAST with a 15 minutes sampling period, and also because traffic analysis from FAST is done on a 15 minutes basis. It should be noted, nonetheless, that the connector can re-sample the data using an arbitrary sampling period.
- Generates a new "Flow" variable (number of cars per hour) from measurements of "Volume" (number of cars per 15 minutes).
- Repairs the data stream, filtering outliers and masking missing data points, using a data imputation algorithm to provide the predictors with a healthy dataset (the data imputation algorithm will be described in detail in the next section).

This service provides a direct way to connect to field traffic data, adding preprocessing capabilities in the edge and offering authentication methods for data accessing, which makes the solution more accessible, functional and secure.

3.2.3.2. Visual Interface

The visual interface is a web page where a map of the Las Vegas I-15 Freeway and its detectors are displayed. The user can interact with each of the detectors to access real time data and traffic predictions calculated by each of the predictors hosted in the cyber layer. The visual interface is depicted in the upper right hand side of Fig. 3.1.

The visual interface is composed by two sub services that run in parallel and communicate with each other by accessing and modifying a group of shared geojson files that contain the data and geolocation of each detector. The first sub service is the "geojson manager", which is in charge of: i) the communication with the Connector service through MQTT messages; and ii) the integration of the data from the FAST platform and the predictors in the geojson files. The second sub service is the "Web Map Interface",

which is in charge of serving the visual interface that delivers to the user the information stored in the geojson files.

3.2.3.3. Predictor Instance

The predictor instance is the core service of the traffic prediction application. There is one predictor instance for each prediction model that is implemented in the system. Each predictor instance is connected to the data streams using MQTT and, after calculating the predictions based on the data, it publishes the predictions via MQTT to be consumed by the visual interface service, which will handle the information and display it on the map. This provides scalability to the solution, since the number of predictor instances that can be hosted in the system is not limited by the architecture, on the contrary, the modular scheme allows implementing a variety of prediction models in parallel to, for example, conduct comparative analyses.

3.3. Datasets

The performance of data-driven prediction models depends heavily on the quality of the datasets used for training, validation and testing. Therefore, in this work we create new datasets, with similar characteristics to those used in state-of-the-art works on traffic prediction but with emphasis on real-world implementation issues, such as seasonal traffic changes, infrastructure modifications and unexpected events. These datasets also constitute a contribution to this research area.

Three datasets were generated using historical data from 2018 and 2019 obtained from the FAST platform. The main characteristics of the datasets are as follows:

- Each dataset has a different purpose: one for training, one for validation and one for testing.

Table 3.1. Description of traffic variables used in the study.

Variable	Range	Unit
Flow	0 - 3975	vehicles/hour
Occupancy	0 - 66.5	percentage vehicles/lane
Speed	0 - 85	miles/hour

- The datasets consider two years of I-15 NorthBound highway traffic data, gathered during 2018 and 2019. With this amount of data, seasonal changes can be discerned.
- Measurements are of vehicle flow, road occupancy and average vehicle speed. Table 3.1 describes the variables used to generate the datasets.
- A sampling period of 15 minutes is inherit from FAST historical database.

To generate the datasets, an initial study of the raw data was conducted to evaluate the richness and correctness of the data. Four main problems were detected, which needed to be addressed.

(i) Invalid measurements.
(ii) Data losses.
(iii) Unevenly sampled data.
(iv) Missing data.

Regarding invalid measurements, about 16% of the data was flagged as invalid or failed by the FAST platform itself (indicative of faulty sensors). Consequently, this data chunk was removed, greatly reducing the amount of available data.

As for data losses, Figure 3.2 shows the available data for each detector in the segment of the I-15 freeway under analysis. It can be seen that some detectors present a extremely high data loss rate, which makes it impossible to use a data imputation technique to repair the time series. Consequently, sensors with less than 50% of the expected data were removed from the study. After the removal, 38 detectors were available, as shown in Figure 3.3. However, there is a large spacial gap between detector 40 and detector 46,

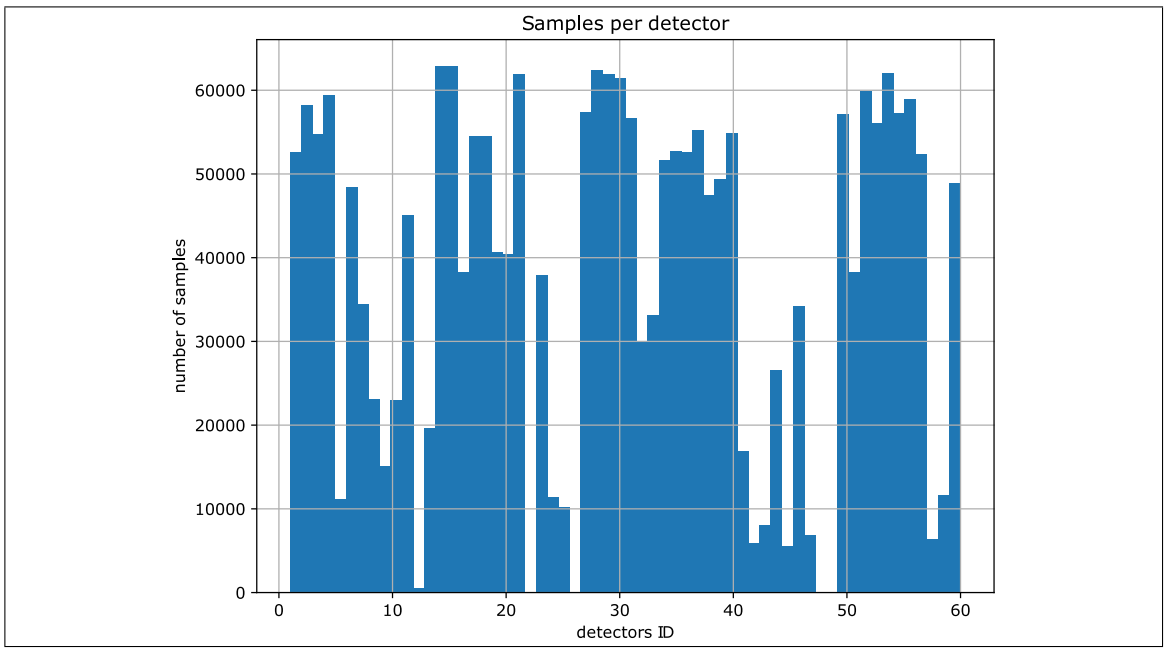


Figure 3.2. Collected data per detector in a two years time frame.

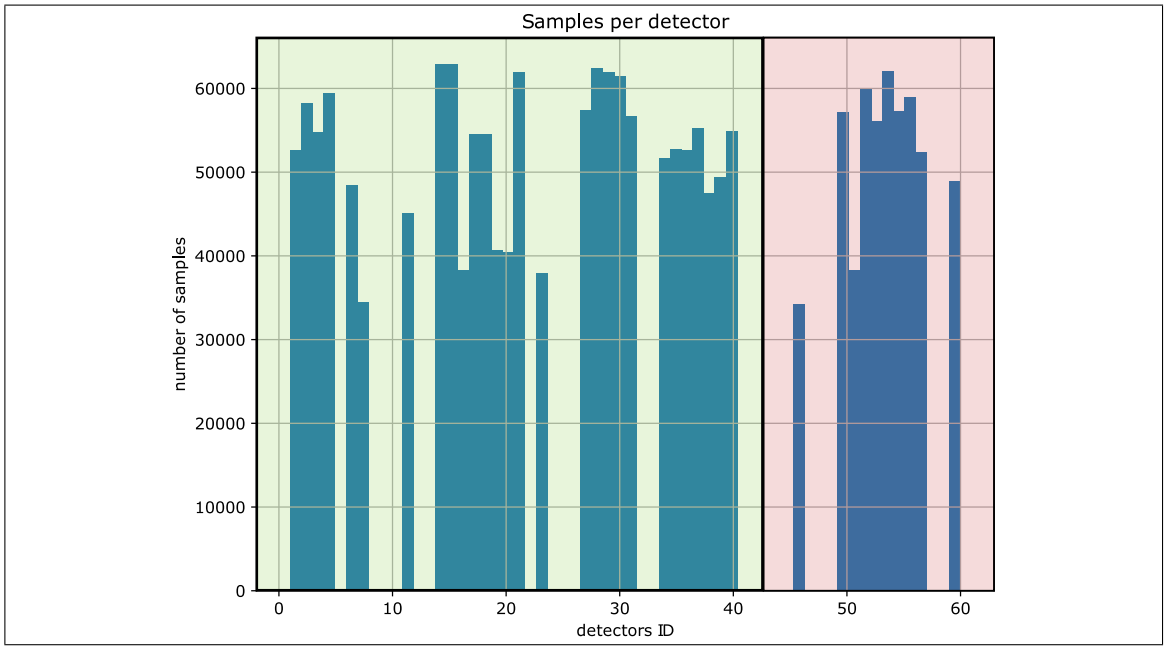


Figure 3.3. Collected data per detector after the removal of detectors with less than 50% of the expected data. The green region contains the final 28 detectors used to generate the datasets, while the red region contains the 10 healthy detectors that were eliminated due to the large spatial gap.

which is geographically located just at the intersection between the I-15 and the US95 highways, an undesirable place to have a blind spot. For this reason, the 10 rightmost detectors in Figure 3.3 were eliminated from the study, finally leaving 28 healthy detectors for generating the datasets.

Regarding unevenly sampled data, due to communication imperfections and data compression algorithms executed by FAST, the sampling period was not constant at 15 minutes. Only about 40% of the data had an elapsed time of 15 minutes between consecutive samples. Although more than 80% of the samples were taken with a sampling period between 14-17 minutes, even a minor jitter in the sampling period produces loss of periodicity in the dataset, which is detrimental for training of data-driven prediction models. To deal with this issue, resampling of the database to a constant sampling period was conducted using an algorithm based on nearest neighbor data and linear interpolation. Briefly, the initial time of the dataset was defined and an allowable temporal error was set. The raw dataset was then walked through from the initial time in 15 minutes steps and the resampled dataset was populated based on the following rules:

(i) Nearest neighbor data if the sample is within the allowable temporal error.
(ii) Linear interpolation if the first condition is not met and both the sample before and after are available.
(iii) NAN value if none of the previous conditions are met.

Finally, to address missing data, a data imputation strategy was developed using the mice-forest library to implement a random forest algorithm, which uses the correlation between the data to fill the missing samples. The strategy used is an iterative algorithm that creates a random forest model for each of the variables that converges to the closest true value at each iteration. The resulting imputation model can be used in real time for imputation of new data. For more details of the algorithm please refer to the library source code Wilson (2020).

Table 3.2. Dataset description

Dataset	Size (samples)	Collection Dates
Train2018	35,040	01/01/2018 - 12/31/2018
Val2019	17,520	01/01/2019 - 06/31/2019
Test2019	17,520	07/01/2019 - 12/31/2019
Retrain2021	4,504	04/16/2021 - 06/02/2021

After solving the aforementioned issues, two years of validated data from 28 detectors was available. The three datasets were generated as follows, the first 50% of the data (one year) was used to create the *Train2018* training dataset, the next 25% to create the *Val2019* validation dataset and the final 25% to create the *Test2019* testing dataset.

Given that two major disruptive events took place since 2019: i) project NEON (largest public works project in Nevada history) that completely reconfigured the I-15 and US95 highway connection; and ii) the COVID outbreak that significantly changed traffic patterns; it was expected that directly applying online the predictors trained with *Train2018* would not give good results. Hence, data was gathered during 2021 to create the *Retrain2021* dataset, which will be used to incrementally retrain the predictors before testing them online on the developed CPS. This dataset consists of 48 days of traffic data with the same measured variables and the same sampling period as before, namely, flow, occupancy, and speed with a sample period of 15 minutes. Table 3.2 shows the main characteristics of the datasets, which are publicly available in Guzmán (2020).

In addition to the datasets, another major output of the initial data analysis process is the imputation model. The final imputation model is used in real-time by the Connector service at the cyber layer for on the fly data repairing. Specifically, the process executed by the connector considers a first gathering stage to resample the data from one sample per minute to one sample every 15 minutes (as mentioned in the previous section), and then a data repairing stage using the imputation model. The flow of the process executed by the connector service is detailed in Figure 3.4.

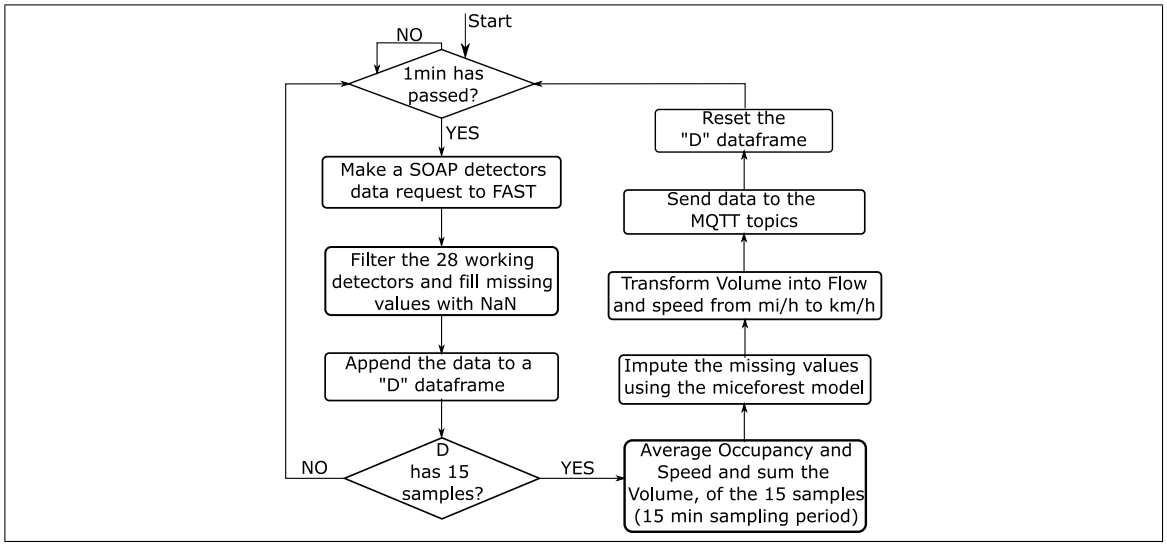


Figure 3.4. Connector service real-time data process diagram.

3.4. Traffic Prediction

For traffic prediction purposes, four different data-driven prediction models were implemented: an eRCNN with linear output, an eRCNN with iterative output, a novel error recurrent encoder decoder (eRED) architecture, and the GWNet (GCNN). Following the common practice in the literature (Wang et al., 2016), vehicle speed is used as the indicator of traffic level; hence, models were trained to predict vehicle speed for different time horizons for each of the detectors. A set of $N = 28$ detectors is considered and a prediction horizon of $W = 4$ time steps is used, yielding a prediction horizon of 60 minutes, considering a sampling period of 15 minutes. Thus, the output of each model is a spatio-temporal image $Io \in R^{N \times W}$ with a single channel, which is the predicted vehicle speed.

The prediction models differ in the structures taken as inputs. The eRED and both types of eRCNN use as input a spatio-temporal image $Ie \in R^{C \times N \times T}$, where the C channels correspond to each of the variables received from the detectors in the FAST platform: Flow, Occupancy and Speed, the second coordinate represents each of the $N = 28$ detectors and the third coordinate contains $T = 24$ samples representing 6 hours of previous

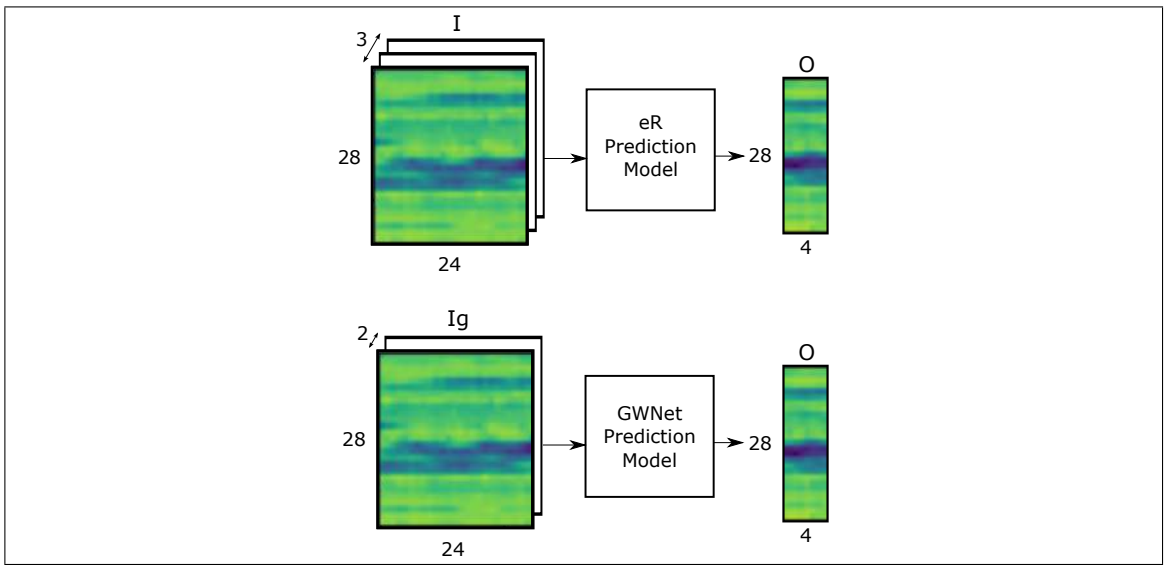


Figure 3.5. Top: Input/output spatio-temporal images of the error recurrent models; Bottom: Input/output spatio-temporal images of the GWNet model

data per detector. On the other hand, the GWNet model uses as input a spatio-temporal image $Ig \in R^{2\times N\times T}$, where one channel is the speed and the other is an index representing the timestamp of the corresponding speed measurement. The rationale is that the models can predict one hour into the future by using the information contained in the last 6 hours of data. A sample of the input and output images can be seen in Fig. 3.5.

The architecture of each of the prediction models is detailed in the following.

3.4.1. eRCNN

The eRCNN models implemented in this work are based on the original eRCNN proposed in Mena-Oreja and Gozalvez (2020). Error recurrence models seek to obtain feedback from recent prediction errors to improve the performance of future predictions. To this end, an error function is defined, whose value is calculated each time a new prediction is made, and then an error vector Ve including the last M values of the error function is used as a recurrent input to the network.

The architecture of the eRCNN model from Mena-Oreja and Gozalvez (2020) is formed by a 2D CNN that receives the spatio-temporal image Ie as input and produces an intermediate output y_{conv} that is passed through a fully connected layer FC_c to generate a hidden state vector h_C. The hidden vector is then concatenated with a feedback hidden state vector h_E, which is obtained from passing the error vector $Ve \in R^{NM}$ through a fully connected layer FC_e. Finally, the resulting hidden vector is passed through a fully connected layer FC_o which outputs the predicted image Io, i.e.,

$$Io = FC_o(concat(FC_c(y_{conv}), FC_e(Ve)). \tag{3.1}$$

The eRCNN was originally designed to predict the speed at a single detector at a given time instant, i.e., $Io \in R$. In order to output a spatio-temporal image, modifications had to be made. To this end, we propose two models based on the original eRCNN. Both models preserve the convolutional part of the original model, but implement error recurrence differently. The first proposal is called eRCNN with iterative output (eRCNNIter). It generates an initial output vector $Io_j \in R^N$, with $j \in [1, W]$, which is then iterated to generate the additional $W - 1$ columns of the prediction image. To do so, Io_1 is obtained in the original way, then $Ve_1 = Ve$ is updated by dropping the oldest element and appending a copy of the last element. Next, the updated error vector is passed through the FC_e layer to obtain h_E, while the initial output Io_1 is passed through a new auxiliary fully connected layer FC_h producing a vector $h_C{}'$, of the same size of h_C. By doing this, two auxiliary hidden state vectors h_E and $h_C{}'$ are created, which are concatenated and passed through the last layer FC_o to produce the next prediction Io_2.

$$Io_1 = FC_o(concat(FC_c(y_{conv}), FC_e(Ve_1)) \tag{3.2}$$

$$Ve_{j+1} = concat(Ve_j[1 : M], Ve_j[M]) \tag{3.3}$$

$$Io_{j+1} = FC_o(concat(FC_h(Io_j), FC_e(Ve_{j+1})) \tag{3.4}$$

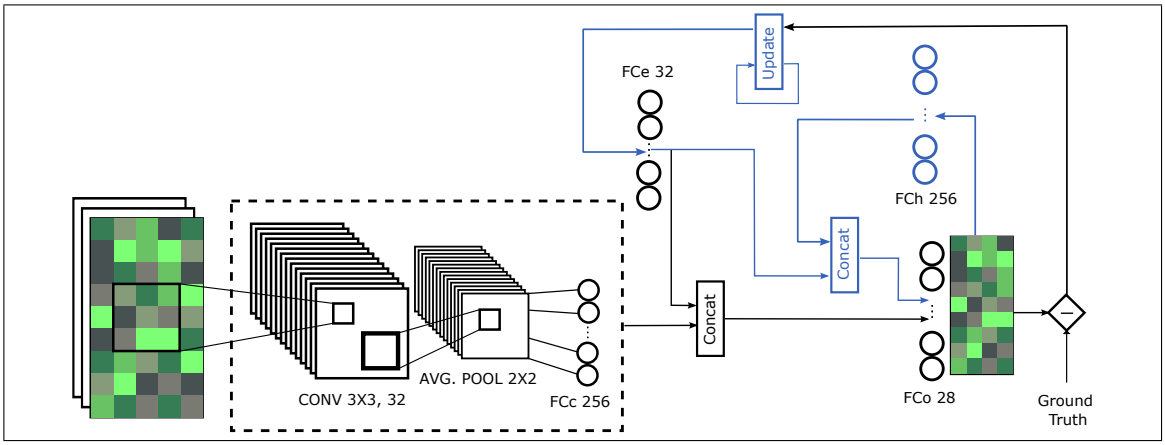


Figure 3.6. Schematic of the eRCNNIter, i.e., eRCNN with iterative output. The blue section represents the part of the model that iterates W-1 times to generate the final output.

This process is repeated $W-1$ times to complete the W prediction window elements, and concatenated to generate the final output image. Fig. 3.6 shows a schematic of the eRCNNIter model.

The second proposal is called eRCNN with linear output (eRCNNLin). This is a slightly simpler model since it maintains the same structure of the original eRCNN, with the difference that the last layer FC_o generates a vector $Io \in R^{NW}$ representing the concatenation of predictions over the prediction horizon. Obtaining a spatio-temporal image from Io is straightforward by resizing. Fig. 3.7 shows a schematic of the eRCNNLin model.

3.4.2. eRED

As second effort, the error recurrence technique is used on a type of encoder-decoder network that has shown good performance in predicting time-series data from industrial applications (Langarica et al., 2020).

Briefly, in its original formulation this model uses a 1D CNN as an encoder, which extracts spatial information from the input variables by generating an encoded vector that

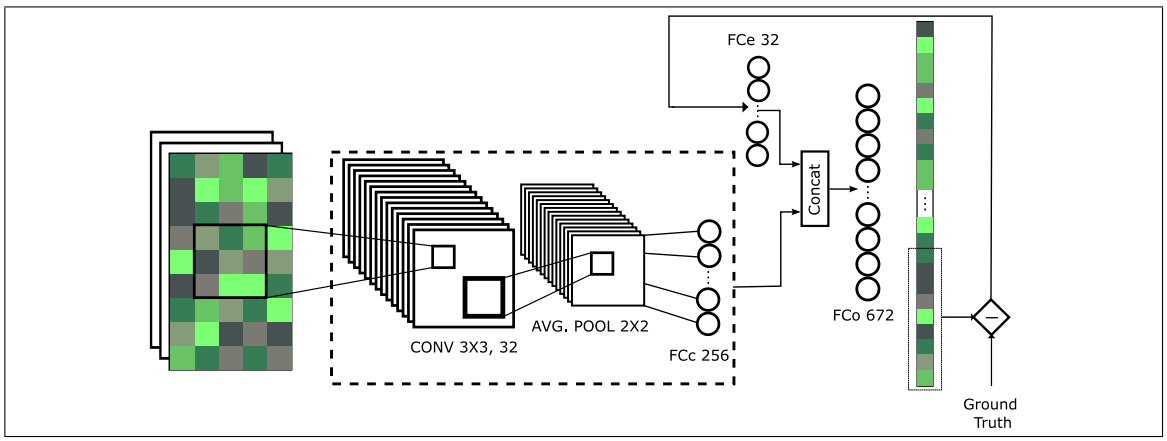


Figure 3.7. Schematic of the eRCNN with linear output which generates all predictions in one step.

is later used as the initial hidden state for a recurrent decoder. The network uses as input a data matrix $X_r(n) \in R^{N \times T_r}$ formed by sequences of length T_r from N variables. The decoder is a GRU-based network with L_R layers and Q_l neurons at layer l, that generates a final hidden state $h_R^{L_R}$ that is passed through a linear layer to obtain the final output. For more information of the original model refer to (Langarica et al., 2020).

In this work, modifications were made to include error recurrence to produce the eRED model. To this end, the encoder was replaced by the 2D CNN used in eRCNN models, which allows extraction of both spatial and temporal information from the input, to generate the initial hidden state h_D0 of the recurrent decoder.

$$h_D0 = FC_c(y_{conv}) \tag{3.5}$$

Unlike the original model, in this case the decoder receives as input the error vector Ve (created with the last M errors values) to generate the output. Hence, after M iterations, $h_r^{L_R}(M) \in R^{Q_{L_R}}$ is available, and the final output Io is obtained by passing this vector through a linear layer. Fig. 3.8 shows a schematic of the eRED model.

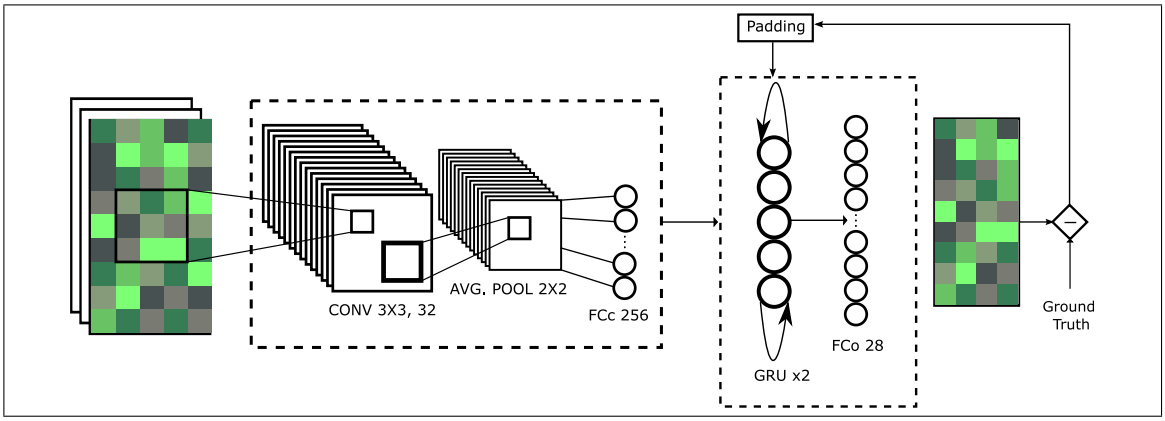


Figure 3.8. Schematic of the error recurrent encoder-decoder eRED model.

3.4.3. GWNet (GCNN)

The last model is the GWNet, which is a GCNN as presented in (Wu et al., 2019). GWNet has shown excellent performance in traffic prediction, outperforming other state-of-the-art DNN models (Yin et al., 2021).

Since GWNet works on graphs, it is necessary to formulate our traffic prediction problem in this context. Graphs are a type of data structure composed of nodes and edges, in which nodes represent objects that have information and edges connect nodes that are related. The advantage of working with graphs is that the information of the relationship between nodes is explicitly used for prediction. Mathematically, a graph G is represented as $G = (V, E)$, where V is the node set and E is the edge set. Each node $v \in V$ and each edge $e \in E$ have features x_v and x_e that change dynamically over time, so at each time step t the graph G has a feature matrix $X(t)$ (Wu et al., 2019).

The general principle of a GNN is to learn neighborhood embeddings by aggregating information from a node neighborhood via edges using neural networks (Liu & Zhou, 2020). In other words, given a graph G and a sequence of s historical feature matrices, the problem is to learn a parametric function f capable of forecasting the graph features T steps into the future.

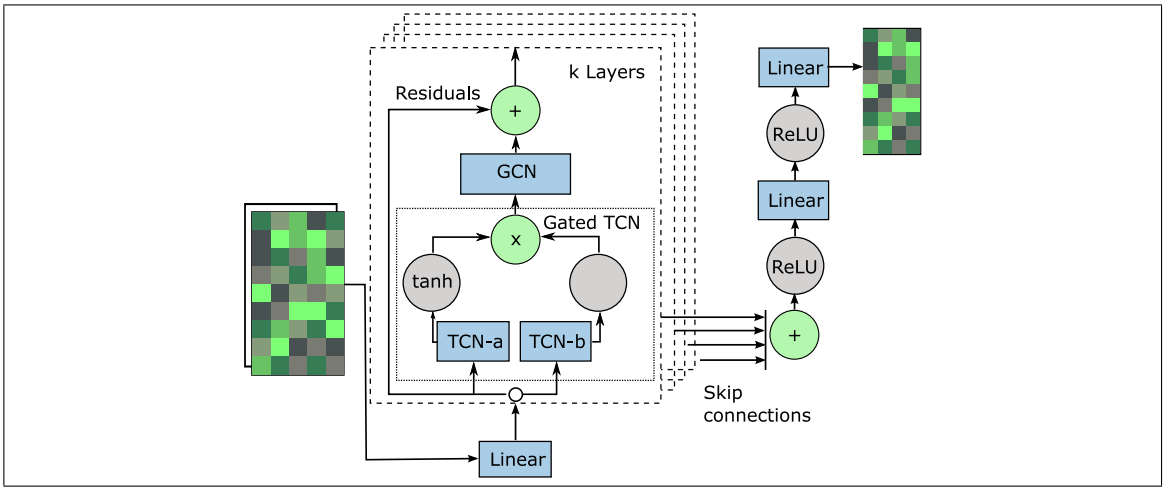


Figure 3.9. Schematic of the GWNet model that explicitly models spatial relationships between detectors with a graph.

GCNN are a subgroup of GNNs that aim to generalize convolutions to the graph domain. These models are the dominant type of GNN and were used for designing the GWNet. For detailed information about the operation of GNNs the reader is referred to Liu and Zhou (2020).

The GWNet is a model consisting of several layers, referred to in Wu et al. (2019) as spatio-temporal layers. Each layer consists of a GCNN, which extracts the spatial information from the data, and a gated temporal convolution layer (Gated TCN), which is formed by two parallel temporal convolution layers (TCN-a and TCN-b), which extract temporal information from the data. In this way, the model is able capture different temporal levels. Finally, the output of each layer is added and passed through a group of linear layers to obtain the final prediction. Fig. 3.9 shows the architecture of the GWNet model.

3.5. Experimental Evaluation

To illustrate the potential of the proposed system, an instance was implemented at the Transportation Research Center (TRC) of the University of Nevada, Las Vegas.

Table 3.3. Experimental setup tools and libraries

Tool	Description	Use Case	URL
Supervisor	A client/server system that allows users to monitor and control a number of processes	Services startup, error management and log generation	`http://supervisord.org/`
Zeep	A fast and modern Python SOAP client	SOAP communication at the Connector service	`https://docs.python-zeep.org/en/master/`
Mosquitto-Server	An open source message broker that implements the MQTT protocol	Run the MQTT Broker for the internal communication with secured authentication	`https://mosquitto.org/`
Paho MQTT	A Python MQTT client library	Run the MQTT client in the services	`https://www.eclipse.org/paho/`
Pytorch	An optimized Python tensor library for deep learning using GPUs and CPUs	Implementation of prediction models	`https://pytorch.org/`
Geopandas	An open source Python library that extends the datatypes used by pandas to allow spatial operations on geometric types	Construction and formatting of the MQTT messages and handling of the ".geojson" files	`https://geopandas.org/`
Flask	A lightweight WSGI web application framework	Implementation of the web interface	`https://palletsprojects.com/p/flask/`
Maptiler	A mapping software platform that allows using global mapping information	Access to I-15 freeway map information	`https://www.maptiler.com/`

3.5.1. Experimental setup

All the services of the CPS were implemented on a server with an Intel Xeon E5-2420 v2 @ 2.20GHz CPU and 32 GB of RAM, running Ubuntu server 16.04.7 as operating system. Each service runs as an individual python script, except for the visual interface, which requires a separate script for each of its sub-services (two scripts in total). Python 3.9.1 was used for the implementation. The tools and libraries used to implement the CPS are detailed in Table 3.3.

It should be noted that, although for this instance everything was installed on the same machine, there is no need to do so. Each service can be deployed in different machines, as long as communication with the MQTT Broker is guaranteed.

Table 3.4. Prediction results. Dataset Test2019

Model	MSE	MAE
eRCNN Iterative	31.61	3.43
eRCNN Linear	19.74	2.47
eRED	24.92	2.89
GWNet	23.79	2.17

For training the predictors, the Adam optimizer with a learning rate of 0.0001 was used considering the mean square error (MSE) as loss function. All the models were initially trained for ten epochs using *Train2018* for predicting vehicle speed over a prediction horizon of 60 minutes, i.e., considering four time steps into the future. *Val2019* was used for validation.

3.5.2. Test2019 results and analysis

To evaluate the performance of the system in predicting vehicle speed, the following strategy was put at work. First, predictors were tested offline using *Test2019*. Good performance is expected since it is assumed that *Train2018*, *Val2019*, and *Test2019* were obtained under similar traffic patterns.

Results in terms of MSE and Mean Absolute Error (MAE) in mph are presented in Table 3.4. It can be seen that, as expected, all the models obtain good results, with an MAE around 5% of the mean value of the predicted variable, vehicle speed. Among the different predictors, the eRCNN with linear output obtained the best performance for the MSE metric, while the GWNet obtained the best performance for the MAE metric. It is worth noting that the eRED model, which is the smallest network, is able to obtain results in the same order of magnitude as the other more complex models.

3.5.3. Online results and analysis

For online testing, the predictors trained with *Train2018* were put online in the CPS as predictor instances. As mentioned in Section 3.3, it is expected that directly applying

the models trained with *Train2018* would not give good results. Therefore, four sets of retrained models were generated using the *Retrain2021* dataset: i) a set that starts with the predictors trained with *Train2018* and considers retraining using the first 25% of the data in *Retrain2021*; ii) a set that starts with the models obtained in i) and is incrementally retrained using the second 25% of the data in *Retrain2021*; iii) a set that starts with the models obtained in ii) and is retrained using the third 25% of the data in *Retrain2021*; and iv) a set that starts with the models obtained in iii) and is retrained with the final 25% of the data in *Retrain2021*. After this process, a total of 20 predictors are available for online evaluation. It should be noted that dividing the *Retrain2021* dataset into four parts is arbitrary and that conducting a sensitivity analysis to find the appropriate amount of data for retraining is recommended.

The CPS was put in operation using the 20 predictor instances detailed above for 75 days, from June 3, 2021 to August 17, 2021. This demonstrates the scalability and reliability of the system for making online predictions in real time. Prediction results are presented in Table 3.5. It can be seen that, as expected, models without retraining perform significantly worse with respect to the results presented in Table 3.4. Nonetheless, retraining helps in recovering the performance for all models. It is particularly interesting to note that there is an important performance improvement after retraining with only 25% of the retraining dataset. This suggests that the predictors can quickly incorporate data with new traffic patterns and recover a good performance. Another important factor to consider is that most predictors keep improving as retraining progresses, implying that conducting periodic incremental retraining is beneficial in a real online implementation.

In terms of performance differences among the predictors, it can be seen that the eRED model, the worst without retraining, greatly improves its performance with retraining, while the eRCNN with iterative output seems to get stuck, which suggests that the eRCNN with iterative output lacks flexibility to adapt to changing traffic patterns hence requiring a large dataset to deliver accurate predictions. The GWNet is undoubtedly superior to the rest in all regards, quickly reaching a performance comparable to the original numbers

Table 3.5. Online prediction results with varying amounts of retraining.

Model	No Retrain		Retrain 25%		Retrain 50%		Retrain 45%		Retrain 100%	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
eRCNN Iterative	100.8	7.47	61.94	5.54	56.72	5.00	59.62	5.12	65.57	5.09
eRCNN Linear	58.22	5.45	39.77	4.01	37.17	3.78	34.80	3.59	34.03	3.53
eRED	133.12	8.62	54.13	4.92	49.96	4.61	47.40	4.35	46.79	4.21
GWNet	44.74	4.2	27.77	2.75	28.17	2.71	27.30	2.71	26.43	2.62

Table 3.6. Online prediction results in terms of the MAE.

Model	No Retrain				Retrain 100%			
	15 min	30 min	45 min	60 min	15 min	30 min	45 min	60 min
eRCNN Iterative	7.76	6.86	7.15	8.11	5.39	4.37	4.98	5.62
eRCNN Linear	4.93	5.07	5.66	6.13	3.31	3.43	3.61	3.77
eRED	8.69	8.59	8.58	8.63	4.08	4.15	4.25	4.39
GWNet	3.40	4.07	4.52	4.84	2.23	2.58	2.76	2.90

presented in Table 3.4. This is consistent with results presented in the literature and reinforces the idea that using a graph-based formulation for traffic prediction is the ideal approach.

A more detailed performance comparison is given in Table 3.6, where performance of the models without retraining and the models retrained with all *Retrain2021* are compared for each of the four prediction steps. It can be seen that, without retraining, the eRCNN with iterative output and the eRED models do not follow the expected trend of decreasing performance as the prediction horizon increases but instead are more consistently poor. This behavior is fixed for the eRED model with retraining. Although the eRCNN with iterative output improves its performance with retraining, it still maintains the erratic behavior. On the other hand, the eRCNN with linear output and the GWNet models show a consistent trend in both cases, with and without retraining, suggesting that these are the most reliable models and hence should be the natural candidates for online traffic prediction, with GWNet model being superior in terms of performance in most cases. Additional analysis of the results can be found in Guzmán (2020).

3.6. Discussion

In this work a CPS for real time data-driven traffic prediction is designed and implemented in the I-15 freeway, Las Vegas urban area. This work is a follow-up to the research presented in chapter 2, and illustrates the type of high-level applications that can be implemented in the cloudlet layer of the proposed architecture for CIM. The system is based on the previously introduced architecture with a physical layer where data is generated, a connectivity layer, a cyber layer where data is processed and is made available in batches every 15 minutes, and a cloud layer where predictions are made and displayed with a visualization tool. A series of deep neural networks were trained as predictors and implemented on the system. Results obtained show the feasibility of performing real time traffic prediction with the available technology deployed on the transportation infrastructure of the Las Vegas urban area and the proposed architecture.

Despite the promising results, Several limitations exist, which will be discussed in the following.

- One of the limitations of this study was the number of detectors used. As mentioned in 3.3, problems of invalid measurements and data losses were encountered when analyzing the data from the I-15 freeway, so the number of detectors studied had to be reduced from 60 to 28 detectors. Part of the data loss is due to a structural change project that was taking place on the highway during the period studied. For this reason, it is proposed to perform a new data collection from the date of completion of the project, in order to reduce the amount of invalid data. In addition, data from other segments covered by the FAST platform could be evaluated, and thus define different segments that meet the amount of valid data needed to train the prediction models.
- A second limitation of the system is that the data is presented with a sampling period of 15 minutes. This limitation is associated to the fact that the data from the Bugatti Fast platform is stored using this sampling period, so historical data

with a higher temporal resolution is not available. However, as of mid-2020, the Transportation Research Lab of the University of Nevada, Las Vegas gained access to data from the FAST detectors at a one-minute resolution. It is therefore proposed to use this new source of information to create a dataset with a one-minute sampling period that will allow us to train and implement models that make predictions every minute.

- As mentioned in 3.2.1, this study was limited to traffic predictions on linear road segments, since spatio-temporal prediction models used in this work have that constraint. However, there is another group of models, specifically from the Graph Neural Networks family, which are capable of using structural and temporal information from networks with more complex morphologies, such as urban areas with intersections and traffic lights. For this reason it is proposed to use the CPS designed in this study together with different GNN models to implement a traffic prediction application in a urban area with intersections.

Closing the chapter, it is worth noting that traffic predictions helps in taking preventive measures to avoid congestion. For this reason, it is a natural idea to use traffic predictions as input data for CIM strategies. Therefore, it looks appealing to use the information encoded by the prediction models in combination with reinforcement learning algorithms to generate a CIM strategy. This idea will be pursued in chapter 4.

4. REINFORCEMENT LEARNING-BASED DISTRIBUTED CONTROL SCHEME FOR INTERSECTION TRAFFIC CONTROL

This Chapter deals with the development of a RL based control scheme that is suitable for being implemented over the conceptual CPS architecture proposed in Chapter 2 and integrates with a version of the traffic predictor detailed in Chapter 3[4]. Therefore, it contributes to the achievement of specific objective (iv).

4.1. Context

As mentioned in Chapter 1, the implementation of Urban Traffic Control (UTC) systems for efficient intersection management has been the standard solution to mitigate congestion. Due to the extensive deployment of traffic detectors and the recent advances in computational equipment to efficiently process data, the current research trend in the area focuses on the use of data-driven algorithms to implement different high-level applications that help in improving traffic conditions (Chen & Englund, 2016). In this setting, two of the most studied applications are: traffic prediction (Guzmán, Morris, & Núñez, 2022), which was elaborated in detail in Chapter 3; and automatic controllers that act in real-time on traffic signals, which allows immediate actions to be taken according to the current traffic conditions.

Due to the data-driven nature of the problem and the stochastic complexity, Neural Networks (NNs) have emerged as a natural tool to implement some of these applications, thanks to their ability to find complex relationships within the process (Yin et al., 2021). Specifically, Reinforcement Learning (RL) NNs models are a promising choice for the design of controllers, given their ability to learn optimal action policies from the interaction with the environment. Hence it is not surprising that an extensive number of works based on RL exist.

[4]The material in this chapter has been submitted for publication to the journal "Guzmán, J., Pizarro, G., & Núñez, F. (2022). A reinforcement learning-based distributed control scheme for cooperative intersection traffic control. Journal of Intelligent Transportation Systems"(Guzmán, Pizarro, & Núñez, 2022).

In Shashi et al. (2021), a long short-term memory (LSTM) based Q-network to decide the action of a traffic light in an intersection is designed. Q-networks are neural networks based on Q learning, a model free reinforcement learning algorithm to learn the value of an action in a particular state. In Shashi et al. (2021), the saturation of the roads, calculated using the Webster method, serves as the representation of the state of the intersections (i.e., the input of the RL network). The control action amounts to choosing the next green phase, relying on the Webster method to calculate the split of the phase. This is done to achieve a stable learning of the RL agent. This approach outperforms the traditional Q-table, a lookup table used to calculate the maximum expected future rewards for action at each state, and then select the max_reward action, in terms of cumulative rewards and average waiting time.

In a different approach, Huo et al. (2020) seeks to improve performance by implementing a cooperative solution to decide control actions. To this end, the state of an intersection is defined as a spatial matrix where each element represents a section of the routes entering and leaving an intersection, and its value indicates whether or not cars are present at that position. Then, all the matrices are grouped into a tensor that, in conjunction with a matrix indicating which phase of each intersection is green, represents the input to a CNN actor-critic. In this case, the output of the network is the probability of switching to the next phase in a predefined cycle, and the control action is defined based on this probability. To stabilize the training, two approaches are taken: on the one hand, the imitation learning technique is used to pre-train the model, making it to imitate an expert controller that seeks to increase the rate of output cars from the intersection vs. input cars; and, on the other, the Proximal Policy Optimization (PPO) algorithm is used to reduce the impact of policy collapse (divergence of the policy) caused by the multi-dimensional output. This model was tested on a nine intersection SUMO environment against a pair of fixed time models and a pair of Q-learning models, using queue size and waiting time as performance indices. The cooperative control model obtained the best results, followed closely by the fixed-time models implemented. Interestingly, Q-learning models presented the worst performance. This work shows that cooperative control helps in improving the

performance of RL controllers but, since in this particular formulation information from all intersections in the system is necessary to make a decision, scalability problems may arise when implementing it in real time.

Considering the observation that CNNs cannot effectively extract the dynamic features of the traffic, like cars directions, Zeng (2021) proposes a decentralized solution based on a GNNs multi-agent advantage actor-critic method, called GraphLight, to act on traffic signals. Since communication between agents is involved when using a GNN as actor-critic, this approach implies a coordinated action making from the agents. Specifically, the state of an agent at time t is a vector containing the current traffic values measured by the detectors of its controlled intersection, combined with the state of neighboring agents multiplied by a spatial discount factor. The control action in this case is the selection of the next green phase that will be active for a fixed period of time d. Testing was done on a 25 intersection SUMO scenario. Results show that the proposed method outperforms state-of-the-art methods in terms of multiple metrics.

Inspired by the success of existing RL-based traffic controllers and by the good results of GNNs for traffic prediction presented in Chapter 3, and following the trend of using more relevant information to understand the state of the road (Chen & Englund, 2016), in this chapter we propose a RL-based distributed control scheme for cooperative intersection control, which works in tandem with a GNN traffic prediction model that encodes the information of intersections and feed the controller. The controller determines the state of the intersection by combining current data from traffic detectors at the intersection with past and future traffic information, from the ego and neighboring intersections, encoded by the GNN traffic prediction model. To illustrate the benefits of the proposed approach, an extensive simulation study is performed, which includes a comparison with classical approaches for intersection management and control.

4.2. Proposed Control Scheme

The proposed control scheme considers that each intersection I_i is associated to an agent a_i that is able to communicate with a set of neighboring agents N_i. The communication between agents enables cooperation, since access to the state information of the neighborhood $V_i = a_i \cup N_i$ will be available to predict the traffic at intersection I_i, as well as to take the control actions. Each intersection is required to be fully instrumented to allow real time traffic data acquisition and control.

For traffic prediction, it is proposed to split the prediction model into a GNN encoder and a linear decoder, so that it would be possible to distribute the encoder part among the agents and the decoder part can be implemented in a cloud layer, in the spirit of the general architecture presented in Guzmán and Núñez (2021). In the proposed scheme, each agent communicates the embedding of the prediction encoder to its neighbors and to the cloud layer; hence, the cloud receives the encoded data from all agents to make the final prediction. By doing so, each agent uses its own traffic data combined with its own and neighboring prediction embedding as inputs to its RL controller.

To take advantage of the predictor structure, a GNN is used for the RL controller and each agent receives the embedding of the controllers of agents in N_i to calculate the control action. Under this setup, control actions are calculated from neighbors' information without the need of querying the cloud. Fig. 4.1 shows a general schematic of the architecture.

4.2.1. Intersection Model

For controller design, the intersection model presented in List and Cetin (2004) (Fig. 4.2) is considered. In this model, the concepts of movements, phases and cycles are used. A *movement* m_j is a flow of cars in a specific direction, and a *phase* $p_k \in P$ is a set of movements that flow simultaneously, within the time interval that this phase is active. As it can be seen in Fig. 4.2, there are eight possible movements ($j \in \{0, 1, \ldots, 7\}$), where the

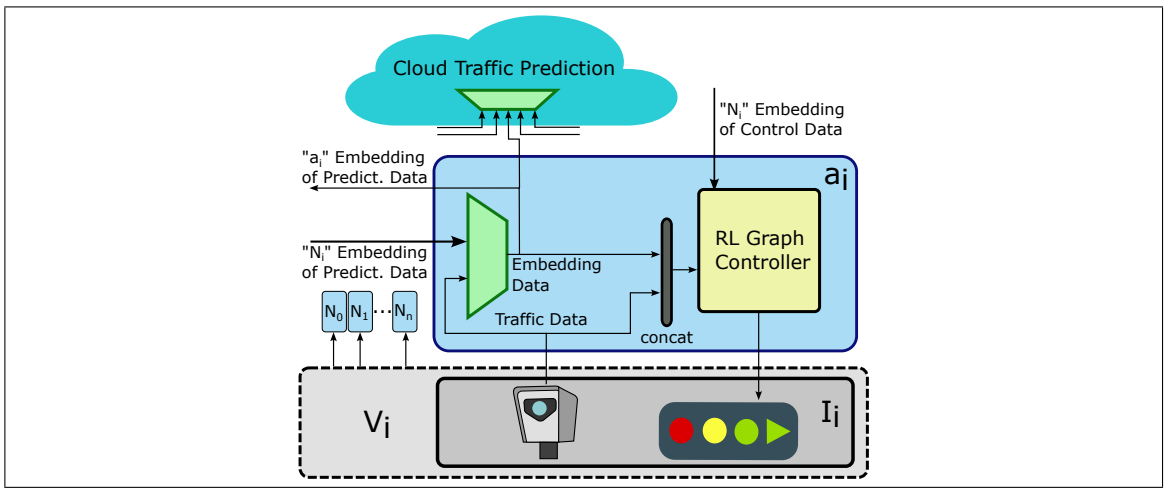


Figure 4.1. General architecture of the proposed RL-based distributed control scheme.

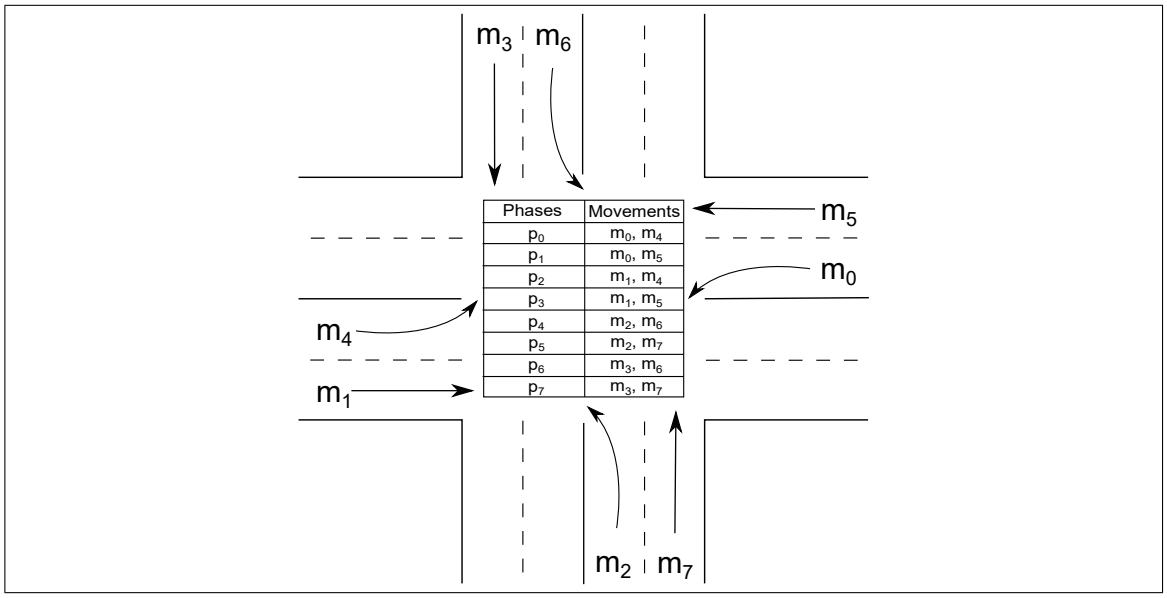


Figure 4.2. Intersection model.

even: m_0, m_2, m_4 and m_6 are left turns, and the odd: m_1, m_3, m_5 and m_7 are through-right combinations. For safety and efficiency, the model defines eight phases ($k \in \{0, 1, \ldots, 7\}$) as pairs of movements that flow without interrupting each other. Only these combinations are allowed. Finally, a cycle C is a defined sequence of phases that is implemented in the traffic light in a cyclic manner.

Table 4.1. Characteristics of measured variables

Variable	Universe	Measure Unit
vehicles density	0 - 1	None
vehicles queue	0 - 1	None
occupancy	0 - 100	percentage
mean speed	0 - max speed	m/s

4.2.2. Instrumentation at the Intersection

It is assumed that each intersection is properly instrumented. Specifically, each intersection is equipped with detectors capable of collecting different traffic variables over a specific area of the incoming roads in real time. The detectors can be, for example, cameras pointing in the direction of the incoming roads that can extract the traffic information using image processing. This way, the sensed area is bounded by the range of the camera. The traffic variables that are assumed available in real time are:

- Vehicle density: Total number of vehicles in the sensed area divided by the approximate number of vehicles that can fit in this area.
- Vehicle queue: Total number of halted vehicles in the sensed area divided by the approximate number of vehicles that can fit in this area.
- Occupancy: Percentage of sensed area occupied by vehicles.
- Mean speed: Mean speed of vehicles in the sensed area.

Table 4.1 describes each of these variables, where max speed = 13.69 m/s. It should be noted that these measured variables are indeed typically available in instrumented traffic infrastructure (Guzmán & Núñez, 2021; Guzmán, Morris, & Núñez, 2022).

4.2.3. Control Action

The controller is a local object of the intersection I_i that configures the activation times t_{p_k} of the phases $p_k \in P_i$ of the fixed traffic light cycle C_i. For simplicity and to add robustness to detector failures that prevent the collection of data from an area, a fixed cycle length Tc_i must be predefined by a theoretical or practical method. Then, starting

from Tc_i, for all the intersections, the controller calculates the optimal green time tG_{p_k} (also called split) for each phase p_k when the cycle starts. The transition times between phases are defined based on values used in the state of the art as three seconds for the yellow time, two seconds for the all red time and five seconds for the minimum split time.

4.3. Traffic Prediction Model

A key feature of the proposed distributed control scheme is the use of future traffic information in the representation of the state of each intersection. To this end, we use the embedding produced by a GNN traffic prediction model. Unlike CNNs, GNNs are usually less complex, have information of the traffic flow direction (reference), and are usually easier to distribute, since each node only needs to have access to its local information and that of a subset of neighboring nodes, hence promoting scalability.

It is necessary to first introduce how graphs and GNNs work to have a better understanding of the proposed strategy. Graphs are a type of data structure composed of nodes and edges, in which nodes represent modules that have information and edges connect nodes that are related. The advantage of working with graphs is that the information of the relationship between nodes is explicitly used. Mathematically, a graph G is represented as a tuple $G = (V, E)$, where V is the node set and E is the edge set. Each node $v \in V$ connects to its neighbors with an edge $e \in E$, and both have features, x_v and x_e, that change dynamically over time. Consequently, at each time step t the graph G has a feature matrix $X(t)$ (Liu & Zhou, 2020). The general principle of the GNN is to learn a neighborhood embedding h_v by aggregating information from the node neighborhood $N(v)$ through a process of "*neural message passing*", in which vector messages are exchanged between nodes and updated using NNs (Hamilton, 2020). This process can be expressed as:

$$o_{N(v)} = aggregate({h_u}^k, \forall u \in N(v)) \tag{4.1}$$

$${h_v}^{k+1} = update({h_v}^k, o_{N(v)}), \tag{4.2}$$

Table 4.2. Movements from neighbors affecting the movements of a nominal intersection.

Active Movement	Input Movements	Output Movements
0	[2, 3, 5] East	[3, 6] South
1	[1, 6, 7] West	[1, 4] East / [3, 6] South
2	[4, 5, 7] South	[0, 5] West
3	[0, 1, 3] North	[3, 6] South / [0, 5] West
4	[1, 6, 7] West	[2, 7] North
5	[2, 3, 5] East	[0, 5] West / [2, 7] North
6	[0, 1, 3] North	[1, 4] East
7	[4, 5, 7] South	[2, 7] North / [1, 4] East

where "$update$" and "$aggregate$" are arbitrary differentiable functions that can be represented by NNs, and $o_{N(v)}$ is the aggregated message from neighborhood $N(v)$. Hence, the inference of a GNN is an iterative process, where at each iteration k the aggregate function takes the set of embeddings from the neighboring nodes $N(v)$ to create a message $o_{N(v)}$ based on the aggregated neighborhood information. Then, in the same iteration, the update function combines the message $o_{N(v)}$ with the previous embedding ${h_v}^k$ to generate the new embedding ${h_v}^{k+1}$. At $k = 0$, the initial embedding ${h_v}^0 = x_v$, i.e., the features of the node v. A common practice is to represent each iteration with a NN layer with its own parameters, so that it is only necessary to stake layers until the desired number of iterations is achieved. Note that, at each iteration, the embedding h_v will include information from more distant neighbors, adding one "hop" per layer, making it a natural option to implement cooperative strategies.

To implement the GNN model, a graph is defined to model the traffic network, where each detector linked to a movement "m_i" represents a node of the graph. The edges are given by the connections with neighboring movements based on the direction of car flow. Table 4.2 shows the input and output connection with the neighboring intersection movements.

In this work, we consider a two layer GNN as traffic prediction model, which is depicted in Fig. 4.3. Using this particular structure is motivated by the results documented

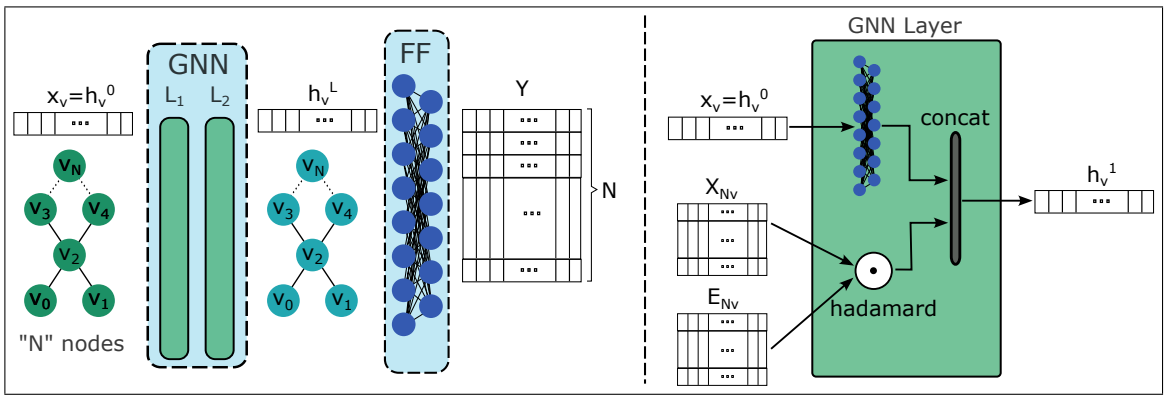


Figure 4.3. Architecture of the GNN network used for traffic prediction and generating the embedding used as input of the RL controller.

in Guzmán, Morris, and Núñez (2022). To create the message $o_{N(v)}$, the $aggregate$ function is the element-wise multiplication between neighbors nodes features and connecting edges features, the latter represented by GNN parameters. The $update$ function consists of passing the previous embedding through a feed-forward (FF) network, and then concatenating the result with the aggregated message $o_{N(v)}$. The embedding h_v generated after the second layer is used as part of the input of the RL controller. Finally, two linear layers are implemented to generate the traffic prediction.

4.4. Reinforcement Learning Controller

The proposed controller is a neural agent in actor-critic architecture based on PPO (Schulman et al., 2017). The fundamentals are exposed in the following.

4.4.1. PPO Algorithm

PPO is an on-policy RL algorithm that optimizes a surrogate objective and maximizes the expected advantage. It is developed from the Trust Region Policy Optimization (TRPO) algorithm (Schulman et al., 2015), where the objective is to find the biggest possible improvement step without stepping so far that accidentally causes a performance collapse. When this is guaranteed, it is possible to perform multiple iterations over the loss

using the same trajectories, without leading to destructively large policy updates. TRPO determines the biggest step sizes using a second-order method, while PPO uses first-order methods that rely on approximations to keep the new policies close to the old ones. There are two variations of PPO, in this work we use PPO-Clip.

To give the context of PPO, the vanilla Policy Gradient (PG) and TRPO are introduced. The PG method works by optimizing the following loss,

$$L^{PG}(\theta) = \hat{\mathbb{E}}[\log \pi_\theta(a_t|s_t)\hat{A}_t], \tag{4.3}$$

where π_θ is a stochastic policy parametrized by θ, $\pi_\theta(a_t|s_t)$ is the probability of taking the action a_t given the state s_t and $\hat{A}_t$ is an estimator of the advantage function at timestep t. The expectation $\hat{\mathbb{E}}[\ldots]$ is taken as the empirical average over a finite batch of samples.

Let $r_t(\theta)$ denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, so $r(\theta_{old}) = 1$ and θ_{old} are the policy parameters before the update. TRPO maximizes the surrogate objective

$$L^{CPI}(\theta) = \hat{\mathbb{E}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right], \tag{4.4}$$

subject to a constraint in the size of the policy updates based on the KL divergence and guarantees monotonic improvement (Schulman et al., 2015).

The superscript CPI refers to conservative policy iteration. Nonetheless, it uses second-order methods that are inefficient. Therefore, PPO modifies the objective by removing the constraint of TRPO and adding a penalization that prevents the policy from moving $r_t(\theta)$ away from 1. Consequently, PPO optimizes the surrogate objective (Schulman et al., 2017)

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t], \tag{4.5}$$

where ϵ is a hyperparameter used to keep the policy updates close to the old one. The motivation for this objective is as follows. The first term inside the $\min$ operator is the surrogate objective from TRPO, L^{CPI}. The second term $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ is used

to remove incentives for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$, which is equivalent to making small changes to the new policy π_θ with respect to the old policy π_θ. Finally, the minimum of the clipped and the unclipped objective is a lower bound on the unclipped objective.

As many other RL algorithms, PPO computes a variance-reduce advantage-function estimator making use of a learned state-value function $V_\phi(s)$ to compute the generalized advantage estimation (GAE). The main idea behind GAE is generating a bias-variance trade-off, which is based on the fact that trajectories close to the present generally have lower variance while variance increases considerably in future trajectories. A general way to present the advantages is through n-step returns given by

$$\hat{A}_t^n = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(s_{t'}, a_{t'}) + \gamma^n V_\phi(s_{t+n}) - V_\phi(s_t), \tag{4.6}$$

where $\gamma < 1$ is the discount factor, and the bigger n used, the higher the variance, while the lower n, the more bias. GAE generalize this advantage estimation by constructing all possible estimators and averaging them, namely,

$$\hat{A}_t^{GAE} = \sum_{n=1}^{\infty} w_n \hat{A}_t^n, \tag{4.7}$$

where $\hat{A}_t^n$ are weighted using an exponential falloff with $w_n \propto \lambda^{n-1}$ and $\lambda < 1$. When (4.7) is expanded, it is simplified into (4.8). In practice, GAE is computed by running the policy for T timesteps, and using the collected samples for an update. This is used to update an estimator that does not look beyond timestep T. Note that PPO's loss (4.5) is defined for any advantage estimation, but in practice uses

$$\hat{A}_t^{GAE} = \sum_{t'=t}^{T} (\gamma\lambda)^{t'-t} \delta_{t'} \tag{4.8}$$

$$\delta_t = r(s_t, a_t) + \gamma V_\phi(s_{t+1}) - V_\phi(s_t). \tag{4.9}$$

The PPO algorithm that uses a fixed-length trajectory is shown in Algorithm 3. Each iteration, there are N actors collecting T timesteps of data each. Then, (4.5) is constructed on these NT timesteps of data and optimized using Adam (Kingma & Ba, 2014).

Algorithm 3 PPO Algorithm.

for iteration = 1,2, . . . **do**
  for actor = 1,2, . . ., N **do**
    Run policy π_{old} in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize L^{CLIP} wrt θ and L^{VF} wrt ϕ for K epochs. $L^{VF} = (V_\theta(s_t) - V_t^{targ})^2$.
  $\theta_{old} \leftarrow \theta$
end for

4.4.2. Actor-Critic Models

In the proposed control scheme, the actor and the critic share structure with each other. Particularly, the actor π_θ and critic V_ϕ share a common feature extractor based on the traffic prediction model GNN. And then, each one projects the embedding vector using a two layer FF network to their respective outputs. The GNN feature extractor embed the observation of each node (given in Table 4.1) and from their neighbors into a hidden vector h_v^L, where L is the number of GNN layers. After h_v^L is available at each node, each node computes the actions and the estimated expected reward given by π_θ and V_ϕ, respectively. It is worth noting that irrespective of the node, all use the same parameters θ and ϕ to compute the learned functions.

The actor outputs a vector $a_{v,t} \in \mathbb{R}^p$, where p is the maximum number of phases available on every intersection. $a_{v,t}$ is a probability vector, i.e., a real-valued vector with non-negative entries that sum 1, containing the proportion that each phase is green during the next traffic light cycle (if the intersection v does not have some phases, the entries of the action vector associated to the non-existing phases are masked). Meanwhile, the critic V_ϕ outputs a scalar containing the estimated expected reward for each node.

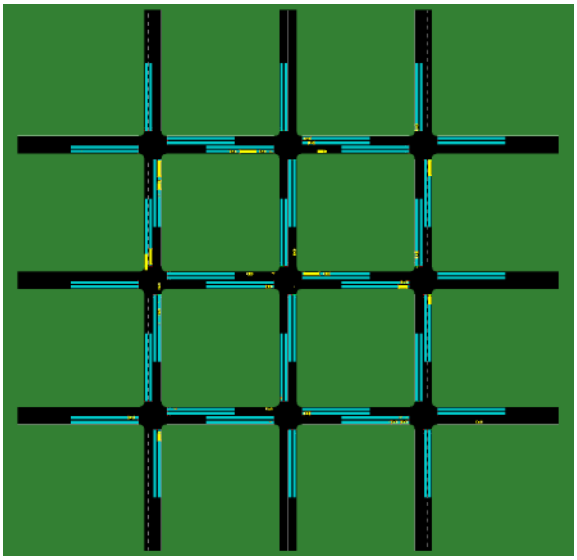


Figure 4.4. Scenario implemented in SUMO.

4.5. Implementation and Experimentation

4.5.1. Test Bed Implementation

A scenario was designed for model testing in SUMO (Krajzewicz et al., 2012). This scenario consists of a grid of nine intersections, where each intersection has eight incoming traffic lines and eight outgoing traffic lines. Each input line represents one of the movements of the model described in Section 4.2, and has a traffic detector that covers an area equal to 50% of the road, i.e., partial observation of the road. In addition, each intersection has a traffic light with a fixed cycle C_i with the following sequence: $p_0 \rightarrow p_3 \rightarrow p_4 \rightarrow p_7$; the split time of each phase can be configured by the controller. Fig. 4.4 shows the scenario implemented in SUMO.

The input data for the traffic prediction model is the last 12 samples of the variable "mean speed" that, with a sample rate of $5min/sample$, represent the mean speed during the past hour. The output are the next four samples, which represent the mean speed of the next 20 min. The embedding h_v for each node v is a vector of size 256. For training, 365 simulations, of roughly one day length each, were performed on the traffic network

scenario. Each simulation was performed with a different traffic level, using a fixed-time controller for traffic light management and employing a normal distribution to choose the traffic level. Area detectors were used with a sample rate of 5 min to generate a dataset of the "mean speed" that represents one year of operation. Finally, the first 70% of the dataset was used for training, the next 10% for validation, and the last 20% for testing.

The RL controller was implemented using the Gym environment. Gym is a standard API for RL that implements a classic "agent-environment loop", where the *agent* makes an *observation* of the state of the environment, and then takes an *action* seeking to maximize a configured *reward* function. In our implementation, an agent is in charge of an intersection. The observation for an agent is an array $O \in R^{m \times s}$, with m equal to the number of input lines (movements) and s the length of the state vector of each line. The state vector is composed by the four traffic variables introduced in Section 4.2, the embedding h_v delivered by the prediction model, and the present split value of the movement. The action $a_{v,t}$ is as introduced in Section 4.4.2, and an observation followed by an action is taken each time a traffic light cycle starts. Several reward functions were evaluated, but the one that gives the best results is the "vehicles difference", defined as

$$rw = vl - vi, \tag{4.10}$$

where vl is the number of vehicles that left the line during green light and vi is the initial number of vehicles in the line before the green light.

Five traffic scenarios Si were created for training and testing the models. It was established that the models should be trained leaving a fixed traffic scenario, because traffic variations made the training unstable and an efficient controller was not reached.

To evaluate the proposed controller, two tests were conducted. First, a sensitivity test of the cycle length, were the RL controller was trained over the scenario S1 (1540 vehicles per hour) using different cycle lengths, so we can find the cycle length that gives the best performance over a fixed traffic scenario. And second, a performance test that seeks to evaluate the controller's capability to adapt to scenarios different from the one learned

during training. In this test, five RL models were trained (one for each scenario) using the best cycle found in the first test, and then were tested against two classical controllers over all the designed traffic scenarios. RL models were named "RL Veh Diff Si", where "i" is the number of the scenario they were trained on.

All models were trained over 1000 epochs, using 28 trajectories for training and 14 trajectories for validation per epoch, where each trajectory consists of one hour of controller submission in the training scenario. Finally, the best-performing models over all epochs were stored for testing. In both tests, cars were introduced during the first hour of simulation, and the simulation finishes either when there are no cars left (all cars have reach their destiny), or after 7200 seconds in case the controller is not able to clear the cars.

4.5.2. Experimental results

For both tests, as performance indices, we propose:

- Waiting time: time in which the cars had a speed lower than 1 m/s.
- Time Loss: Average time lost due to driving below the ideal speed (slowdowns due to intersections, etc.)

For testing the sensitivity to the cycle length, the controllers were trained and tested using five different cycle lengths, $Tc_i \in [40, 50, 60, 70, 80] seconds$. Results are presented in Fig. 4.5. For $Tc_i = 40 seconds$ a efficient controller was never reached, so this result was not plotted. It can be seen that both performance indices find a minimum in $Tc_i = 60 seconds \implies$ **"best performance cycle length"**, and then it starts to deteriorate linearly both as this number increases and decreases. For this reason, this cycle length was used to train all the models of the next test.

For the performance test, motivated by the results in Huo et al. (2020), which proposes a similar objective of cooperative intersection control with RL models, we compare the performance of our controllers with a fixed-time strategy, since this strategy was the closest

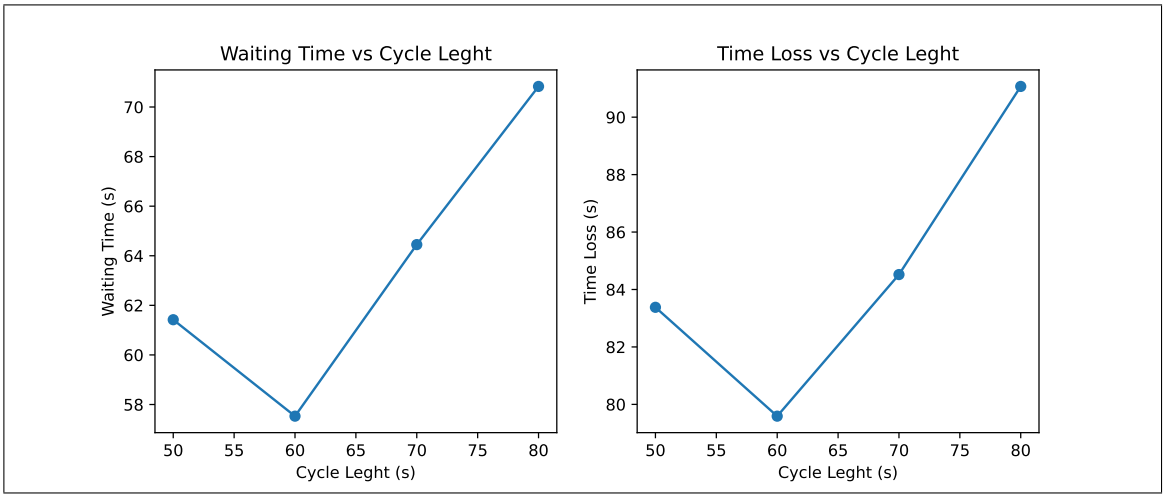


Figure 4.5. Cycle sensitivity test of the "RL Veh Left" controller evaluated in the traffic scenario S1 = 1540 vehicles per hour. The lower, the better

competitor in Huo et al. (2020). This controller has a fixed cycle time, equal in length to the RL controller, but divided equally among the phases. To increase the meaningfulness of the test, it was decided to also compare against the Webster method, which is an optimal control strategy widely used in the real world. A Webster controller was designed for each traffic scenario Si, and to be fair over the test conditions, it was decided to constrain the cycle length of the Webster controllers to 60 seconds, so only the phases split length was configured.

As mentioned before, models were tested over five different traffic scenarios. Each scenario has a different traffic density, but maintains the same traffic distribution. This was done to improve the stability of the training process. For evaluation purposes, each controller was run ten times on the same scenario, and the results reported in Table 4.3 correspond to the average. It can be observed that the *RL Veh Diff Si* controllers obtained the best performance in terms of both performance indices in the specific scenario they were trained on. It can also be seen that each controller performs close to the Webster controller in the traffic scenarios. This demonstrates the capability of the RL controller to handle similar traffic density scenarios to those in which it was trained.

Table 4.3. Results of the evaluated controllers for different traffic scenarios.

Model	Vehicles per Hour				
	1320 (S0)	1540 (S1)	1769 (S2)	1980 (S3)	2200 (S4)
	Waiting Time				
Fixed Time	74.53	82.39	-	-	-
Webster	61.74	63.00	66.33	67.92	75.07
RL Veh Diff S0	**56.73**	64.55	72.40	81.88	106.66
RL Veh Left S1	61.01	**57.53**	67.21	71.75	79.76
RL Veh Left S2	62.10	62.98	**63.11**	71.93	81.78
RL Veh Left S3	60.40	60.60	65.68	**65.93**	74.77
RL Veh Left S4	63.18	63.94	68.43	69.11	**72.90**
	Time Loss				
Fixed Time	95.91	105.73	-	-	-
Webster	81.76	83.48	87.86	90.33	98.83
RL Veh Diff S0	**76.26**	85.59	94.86	106.54	133.97
RL Veh Left S1	81.01	**77.74**	88.93	95.10	104.65
RL Veh Left S2	82.14	83.73	**84.38**	95.32	106.93
RL Veh Left S3	80.11	80.92	87.31	**88.50**	98.78
RL Veh Left S4	83.54	84.83	90.42	92.15	**96.83**

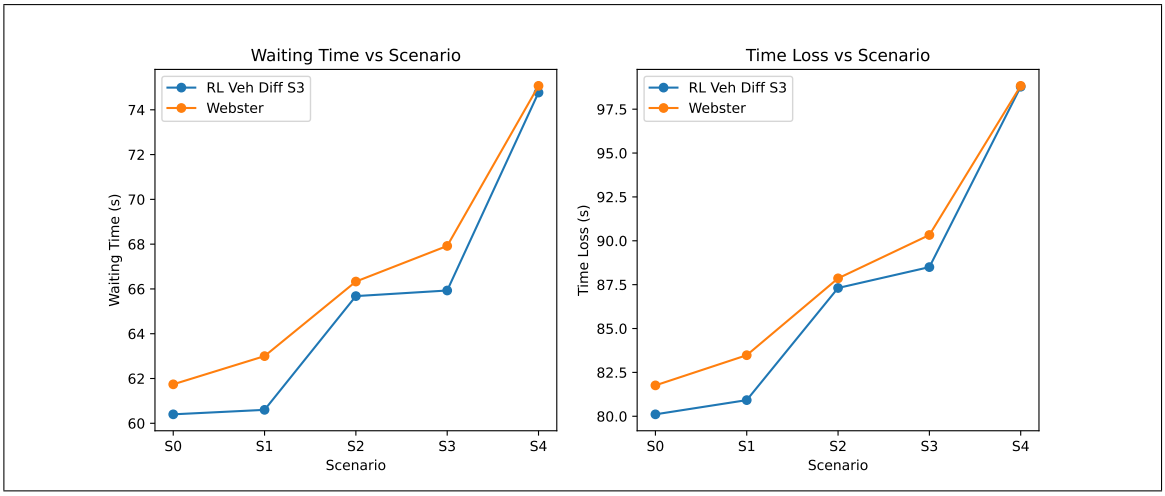


Figure 4.6. Comparison of Waiting Time and Time Loss indices between the RL Veh Diff S3 controller and the Webster controller. The lower, the better

When the controllers face a scenario different from the one they were trained on, it can be seen that the performance of the RL controllers starts to deteriorate compared to the Webster controllers. However, this behavior is slow and less significant when they move to lower congested scenarios, so it is possible to find a specific train scenario that makes

our controller perform better that Webster in a wide traffic density spectrum. This is the case of the "RL Veh Left S3", which performs better than Webster in all the test scenarios. Fig. 4.6 shows the described behaviour.

4.6. Discussion

A novel RL-based distributed control scheme to address the intersection traffic control problem is proposed. The key ingredient is the use of an embedding from a GNN traffic prediction model to represent part of the intersection state, giving the solution a perspective of the future traffic level in the neighborhood. This system can be deployed using the CPS architecture for CIM proposed in Chapter 2, by distributing the RL model and the traffic predictor encoder among the CPOs and implementing the predictor decoder in the cloudlet layer. Simulation results obtained in SUMO show that the proposed controller outperforms the fixed-time and webster standard models in the traffic scenario they were trained. Results also show that models trained in high traffic conditions have also good performance when traffic conditions are close or better to those faced during training.

Despite the promising results, Several limitations exist, which will be discussed in the following.

- An important aspect of this work is that both the training and testing were performed maintaining a constant input traffic distribution as mentioned in 4.5.2. This means that, although several scenarios with different traffic densities were evaluated, the spatial distribution with which cars entered the network was almost always the same. This was done to increase the stability of the training process, but is a clear limitation for the implementation in a real environment. It is therefore proposed to perform a more complex training process, where scenarios with different traffic densities and distributions are gradually explored.
- A second limitation of the research is that the training and testing of the models was performed in an ideal scenario consisting of a grid of fully instrumented

intersections of equal morphologies. This is an ideal scenario very difficult to find in the real world. It is therefore proposed to perform simulation tests using scenarios that are closer to reality, with intersections with different morphologies and not instrumented. It should be taken into consideration that this could bring instability problems in the training process, due to the incorporation of uncertainty in the case of non-instrumented intersections.

5. CONCLUSIONS AND FUTURE WORK

5.1. Concluding Remarks

In this thesis a CPS approach to CIM is proposed, which aims at providing a general architecture that fulfills the functional requirements to address urban traffic control problems. The proposed system considers three layers, namely, i) a physical layer that assumes instrumented intersections and a flexible traffic signal system capable of receiving commands from a cyber entity; ii) a modular cyber layer where local computing elements abstracts the intersection using timed Petri nets for traffic signal control, a data transformation module for congestion modeling and a supervisor for split calculation and communication with the neighbors and superior nodes; and iii) a cloudlet layer for the implementation of high level applications, such as a traffic predictor.

As a first testing pilot, an instance of the proposed architecture was implemented using fuzzy strategies for the congestion modeling and split calculation. Evaluation of the instance in both an ideal scenario, with fully instrumented and identical intersections, and a real scenario, covering a portion of the city of San Diego, validates: i) the functionality of the proposed architecture, since it is capable of handling the generated data and the associated communication and processing load; and ii) its performance, since it increases the efficiency of the traffic by properly controlling the split values at each intersection, outperforming timed, Webster and coordinated control methods.

As a proof of concept of high level applications that can be deployed in the cloudlet layer, a high-level broad-scope data-driven application was designed and implemented, focusing on real time data-driven traffic prediction on the I-15 freeway in Las Vegas urban area. The system is based on the previously introduced architecture with a field layer where data is generated, a connectivity layer, and a cyber layer where data is processed and is made available through a visualization tool. A series of DNNs were trained as predictors and implemented on the system. Online evaluation showed that the GWNet outperforms other models in predicting vehicle speed over a prediction horizon of 60 minutes. Results

obtained show the feasibility of performing real time traffic prediction using the proposed architecture with the available technology deployed on the transportation infrastructure of the Las Vegas urban area.

Finally, to improve the management of the intersection traffic control on the cyber layer of the proposed architecture, a local-scope CIM data-driven application based on a novel RL-based distributed control scheme was designed. The key ingredient of this model is the use of an embedding from a GNN traffic prediction model to represent part of the intersection state, giving the solution a perspective of the future neighborhood traffic level. Simulation results obtained in SUMO show that the proposed controller outperforms the fixed-time and Webster standard methods in the scenarios they are trained on. It was also found that it is possible to outperform these standard methods over a wide traffic density spectrum if the RL model is trained in a proper scenario.

5.2. Directions for Future Research

The results in this thesis indicate that a CPSs approach to CIM is not only feasible, but also advantageous. In order to unleash the full potential of the methodology, several concrete actions can be listed as future work, namely,

- Implementing of a full-scale system in a laboratory environment, following a hardware-in-the-loop scheme, distributing the RL controller and the predictor model encoder among the nodes of the cyber layer of the architecture.
- Increasing the complexity of the GNN models used for both traffic prediction and the RL controller. This can be done by increasing the size and the number of hidden layers, or by using a model that allows extracting space-time information from the network, such as, for example, a distributed GWnet-based model.
- Designing and implementing a control strategy that considers the pedestrian flow into the traffic signal management, by designing a DTM that uses this data to generate the intersection state variable.

- Implementing an encryption system on communication messages between nodes and between system layers, to meet the security requirements necessary for its implementation in a real environment over public Internet connections.
- Conducting scalability tests of the architecture by increasing the number of CPOs to a number comparable to the number of intersections in a city. This will allow detecting and solving communication and real-time response problems that may exist at a larger scale.
- Implementing the timed-petri-net actuator in C language on a microcontroller to validate its performance and to verify the feasibility of its implementation in a real environment.
- Deploying a real intersection traffic control pilot in a controlled environment.
- Implementing a recommendation system at the cloudlet layer in charge of hyperparameter optimization, such as the cycle length of the traffic signals.
- Designing an automatic retraining procedure for all data-driven models, to increase the adaptability of the solution to long term changes on the traffic network, e.g., structural changes.
- Deriving theoretical results regarding the stability and optimality of the proposed collaborative strategy.

REFERENCES

Abdelhameed, M. M., Abdelaziz, M., Hammad, S., & Shehata, O. M. (2014a). Development and evaluation of a multi-agent autonomous vehicles intersection control system. In *2014 international conference on engineering and technology (icet)* (p. 1-6). doi: 10.1109/ICEngTechnol.2014.7016754

Abdelhameed, M. M., Abdelaziz, M., Hammad, S., & Shehata, O. M. (2014b). A hybrid fuzzy-genetic controller for a multi-agent intersection control system. In *2014 international conference on engineering and technology (icet)* (p. 1-6). doi: 10.1109/ICEngTechnol.2014.7016755

Bichiou, Y., & Rakha, H. A. (2018). Developing an Optimal Intersection Control System for Automated Connected Vehicles. *IEEE Trans. Intell. Transp. Syst.*, *20*(5), 1908–1916. doi: 10.1109/TITS.2018.2850335

Brilon, W., & Wietholt, T. (2013). Experiences with Adaptive Signal Control in Germany. *Transportation Research Record*, *2356*(1), 9-16. doi: 10.1177/0361198113235600102

Buzachis, A., Celesti, A., Galletta, A., Galletta, A., Fazio, M., & Villari, M. (2019). A secure and dependable multi-Agent autonomous intersection management (MA-AIM) system leveraging blockchain facilities. *Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018*, 189–194. doi: 10.1109/UCC-Companion.2018.00060

Chen, L., & Englund, C. (2016). Cooperative Intersection Management: A Survey. *IEEE Trans. Intell. Transp. Syst.*, *17*(2), 570–586. doi: 10.1109/TITS.2015.2471812

Comlan, M., Delfieu, D., Sogbohossou, M., & Vianou, A. (2017). Embedding time Petri nets. *2017 4th International Conference on Control, Decision and Information Technologies, CoDIT 2017*, *2017-Janua*(May 2018), 404–409. doi: 10.1109/CoDIT.2017.8102625

Cookson, G. (2018). *Inrix Global Traffic Scorecard 2017* (Tech. Rep.). INRIX Research.

Gregoire, J., Qian, X., Frazzoli, E., De La Fortelle, A., & Wongpiromsarn, T. (2015). Capacity-aware backpressure traffic signal control. *IEEE Transactions on Control of Network Systems*, *2*(2), 164–173. doi: 10.1109/TCNS.2014.2378871

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645-1660.

Guerrero-Ibáñez, A., Contreras-Castillo, J., Buenrostro, R., Barba Marti, A., & Reyes Muñoz, A. (2010). A policy-based multi-agent management approach for intelligent traffic-light control. *IEEE Intelligent Vehicles Symposium, Proceedings*, 694–699. doi: 10.1109/IVS.2010.5548133

Guzmán, J. (2019). *Traffic simulation.* Retrieved from `https://github.com/josegg05`

Guzmán, J. (2020). *CPS Real-time Traffic Prediction.* Retrieved from `https://github.com/josegg05/Vegas_I15_TP`

Guzmán, J., Morris, B., & Núñez, F. (2022). A cyber-physical system for data-driven real-time traffic prediction in las vegas i-15 freeway. *IEEE Intelligent Transportation Systems Magazine*. doi: 10.1109/MITS.2022.3211996

Guzmán, J., & Núñez, F. (2021). A cyber-physical systems approach to collaborative intersection management and control. *IEEE Access*, *9*, 99617-99632.

Guzmán, J., Pizarro, G., & Núñez, F. (2022). A reinforcement learning-based distributed control scheme for cooperative intersection traffic control. *Journal of Intelligent Transportation Systems*, *Under Review*.

Hamilton. (2020). Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *14*(3), 1-159.

Hamilton, Waterson, B., Cherrett, T., Robinson, A., & Snell, I. (2013). The evolution of urban traffic control: changing policy and technology. *Transportation Planning and Technology*, *36*(1), 24–43. doi: 10.1080/03081060.2012.745318

Harrison, R., Vera, D., & Ahmad, B. (2016). Engineering Methods and Tools for Cyber–Physical Automation Systems. *Proc. IEEE*, *104*(5), 973-985. doi:

10.1109/JPROC.2015.2510665

Hirankitti, V., & Krohkaew, J. (2007). An Agent Approach for Intelligent Traffic-Light Control. *Proceedings - 1st Asia International Conference on Modelling and Simulation: Asia Modelling Symposium 2007, AMS 2007*, 496–501. doi: 10.1109/AMS.2007.11

Huang, Y. S., Weng, Y. S., & Zhou, M. (2015). Design of traffic safety control systems for emergency vehicle preemption using timed petri nets. *IEEE Trans. Intell. Transp. Syst.*, *16*(4), 2113–2120. doi: 10.1109/TITS.2015.2395419

Huo, Y., Tao, Q., & Hu, J. (2020). Cooperative control for multi-intersection traffic signal based on deep reinforcement learning and imitation learning. *IEEE Access*, *8*, 199573-199585.

Jin, Q., Wu, G., Boriboonsomsin, K., & Barth, M. (2012). Advanced intersection management for connected vehicles using a multi-agent systems approach. *IEEE Intelligent Vehicles Symposium, Proceedings*, 932–937. doi: 10.1109/IVS.2012.6232287

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krajzewicz, D., Erdmann, J., Behrisch, M., & Bieker, L. (2012). Recent Development and Applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, *5*(3&4), 128–138.

Lana, I., Del Ser, J., Velez, M., & Vlahogianni, E. I. (2018). Road traffic forecasting: Recent advances and new challenges. *IEEE Intelligent Transportation Systems Magazine*, *10*(2), 93-109.

Langarica, S., Pizarro, G., Poblete, P. M., Radrigán, F., Pereda, J., Rodriguez, J., & Núñez, F. (2020). Denoising and voltage estimation in modular multilevel converters using deep neural-networks. *IEEE Access*, *8*, 207973-207981. doi: 10.1109/ACCESS.2020.3038552

Langarica, S., Rüffelmacher, C., & Núñez, F. (2020). An Industrial Internet Application for Real-Time Fault Diagnosis in Industrial Motors. *IEEE Trans. Autom. Sci. Eng.*, *17*(1), 284-295.

Li, Y., Yu, R., Shahabi, C., & Liu, Y. (2018). Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International conference on learning representations.* Retrieved from `https://openreview.net/forum?id=SJiHXGWAZ`

Lin, S., De Schutter, B., Xi, Y., & Hellendoorn, H. (2013). Integrated urban traffic control for the reduction of travel delays and emissions. *IEEE Trans. Intell. Transp. Syst.*, *14*(4), 1609–1619. doi: 10.1109/TITS.2013.2263843

List, G. F., & Cetin, M. (2004). Modeling traffic signal control using Petri nets. *IEEE Trans. Intell. Transp. Syst.*, *5*(3), 177–187. doi: 10.1109/TITS.2004.833763

Liu, Z., & Zhou, J. (2020). *Introduction to graph neural networks* (1st ed.). Morgan & Claypool. doi: 10.2200/S00980ED1V01Y202001AIM045

Ma, X., Dai, Z., He, Z., Ma, J., Wang, Y., & Wang, Y. (2017, 04). Learning traffic as images: A deep convolutional neural network for large-scale transportation network speed prediction. *Sensors*, *17*, 818. doi: 10.3390/s17040818

Marwedel, P., & Engel, M. (2016). Cyber-Physical Systems: Opportunities, Challenges and (Some) Solutions. In A. Guerrieri, V. Loscri, A. Rovella, & G. Fortino (Eds.), *Management of cyber physical objects in the future internet of things: Methods, architectures and applications* (pp. 1–30). Cham: Springer International Publishing.

Mauro, V., & Taranto, C. (1990). UTOPIA. *IFAC Proceedings Volumes*, *23*(2), 245–252. doi: https://doi.org/10.1016/S1474-6670(17)52678-6

Mena-Oreja, J., & Gozalvez, J. (2020). A comprehensive evaluation of deep learning-based techniques for traffic prediction. *IEEE Access*, *8*, 91188-91212. doi: 10.1109/ACCESS.2020.2994415

Mirchandani, P., & Head, L. (2001). A real-time traffic signal control system: architecture, algorithms, and analysis. *Transportation Research Part C: Emerging Technologies*, *9*(6), 415–432. doi: 10.1016/S0968-090X(00)00047-4

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proc. IEEE*, *77*(4), 541-580. doi: 10.1109/5.24143

Núñez, F., Langarica, S., Díaz, P., Torres, M., & Salas, J. C. (2020). Neural Network-Based Model Predictive Control of a Paste Thickener Over an Industrial Internet Platform. *IEEE Trans. Industr. Inform.*, *16*(4), 2859-2867.

OpenStreetMap contributors. (2017). *Planet dump.* Retrieved from `https://www.openstreetmap.org`

Pishue, B. (2021). *Inrix Global Traffic Scorecard 2021* (Tech. Rep.). INRIX Research.

Pommereau, F. (2015). SNAKES: A Flexible High-Level Petri Nets Library (Tool Paper). In R. Devillers & A. Valmari (Eds.), *Application and theory of petri nets and concurrency* (pp. 254–265). Springer International Publishing.

Qi, L., Zhou, M., & Luan, W. (2016). Emergency traffic-light control system design for intersections subject to accidents. *IEEE Trans. Intell. Transp. Syst.*, *17*(1), 170–183. doi: 10.1109/TITS.2015.2466073

Qi, L., Zhou, M. C., & Luan, W. J. (2018). A Two-level Traffic Light Control Strategy for Preventing Incident-Based Urban Traffic Congestion. *IEEE Trans. Intell. Transp. Syst.*, *19*(1), 13–24. doi: 10.1109/TITS.2016.2625324

Reed, T. (2020, 03). *Inrix global traffic raking* (Tech. Rep.). INRIX Research.

Robertson, D. I., & Bretherton, R. D. (1991). Optimizing Networks of Traffic Signals in Real Time—The SCOOT Method. *IEEE Trans. Veh. Technol.*, *40*(1), 11–15. doi: 10.1109/25.69966

Rodríguez, Y., Alejo, C., Alejo, I., & Viguria, A. (2016). ANIMO, Framework to Simplify the Real-Time Distributed Communication. In A. Guerrieri, V. Loscri, A. Rovella, & G. Fortino (Eds.), *Management of cyber physical objects in the future internet of things: Methods, architectures and applications* (pp. 77–92). Cham: Springer International Publishing.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning* (pp. 1889–1897).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shashi, F. I., Md Sultan, S., Khatun, A., Sultana, T., & Alam, T. (2021). A study on deep

reinforcement learning based traffic signal control for mitigating traffic congestion. In *2021 ieee 3rd eurasia conference on biomedical engineering, healthcare and sustainability (ecbios)* (p. 288-291).

Shirazi, M. S., & Morris, B. T. (2017). Looking at Intersections: A Survey of Intersection Monitoring, Behavior and Safety Analysis of Recent Studies. *IEEE Trans. Intell. Transp. Syst.*, *18*(1), 4-24. doi: 10.1109/TITS.2016.2568920

Sims, A., & Dobinson, K. (1980). The Sydney coordinated adaptive traffic (SCAT) system philosophy and benefits. *IEEE Transactions on Vehicular Technology*, *29*(2), 130-137. doi: 10.1109/T-VT.1980.23833

Tang, B., Chen, Z., Hefferman, G., Pei, S., Wei, T., He, H., & Yang, Q. (2017). Incorporating Intelligence in Fog Computing for Big Data Analysis in Smart Cities. *IEEE Trans. Industr. Inform.*, *13*(5), 2140-2150. doi: 10.1109/TII.2017.2679740

United Nations. (2018). *World Urbanization Prospects: The 2018 Revision* (Tech. Rep.). Department of Economic and Social Affairs, Population Division.

UTMC Ltd. (2009). UTMC Framework Technical Specification (3rd ed.) [Computer software manual].

Wang, J. (1998). *Timed Petri nets: theory and application*. Springer US. doi: 10.1007/978-1-4615-5537-7

Wang, J., Gu, Q., Wu, J., Liu, G., & Xiong, Z. (2016). Traffic speed prediction and congestion source exploration: A deep learning method. In *2016 ieee 16th international conference on data mining (icdm)* (p. 499-508). doi: 10.1109/ICDM.2016.0061

Warner, J., & Sexauer, J. (2019). *JDWarner/scikit-fuzzy: Scikit-Fuzzy version 0.4.2.* Zenodo. Retrieved from `https://doi.org/10.5281/zenodo.3541386` doi: 10.5281/zenodo.3541386

Wilson, S. (2020). *miceforest.* Retrieved from `https://github.com/AnotherSamWilson/miceforest`

Wu, Z., Pan, S., Long, G., Jiang, J., & Zhang, C. (2019). Graph wavenet for deep spatial-temporal graph modeling. In S. Kraus (Ed.), *Proceedings of the twenty-eighth international joint conference on artificial intelligence, IJCAI 2019, macao,*

china, august 10-16, 2019 (pp. 1907–1913). ijcai.org. Retrieved from https://doi.org/10.24963/ijcai.2019/264 doi: 10.24963/ijcai.2019/264

Xu, B., Ban, X. J., Bian, Y., Li, W., Wang, J., Li, S. E., & Li, K. (2019). Cooperative Method of Traffic Signal Optimization and Speed Control of Connected Vehicles at Isolated Intersections. *IEEE Trans. Intell. Transp. Syst.*, *20*(4), 1390–1403. doi: 10.1109/TITS.2018.2849029

Ye, F., Qian, Y., Hu, R. Q., & Das, S. K. (2015). Reliable Energy-Efficient Uplink Transmission for Neighborhood Area Networks in Smart Grid. *IEEE Transactions on Smart Grid*, *6*(5), 2179-2188. doi: 10.1109/TSG.2015.2392130

Yin, X., Wu, G., Wei, J., Shen, Y., Qi, H., & Yin, B. (2021). Deep learning on traffic prediction: Methods, analysis and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 1-17. doi: 10.1109/TITS.2021.3054840

Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014). Internet of Things for Smart Cities. *IEEE Internet of Things Journal*, *1*(1), 22-32. doi: 10.1109/JIOT.2014.2306328

Zeng, Z. (2021). Graphlight: Graph-based reinforcement learning for traffic signal control. In *2021 ieee 6th international conference on computer and communication systems (icccs)* (p. 645-650).

Zhang, X., & Riedel, T. (2017). Urban traffic control: present and the future. *International Journal of Urban Sciences*, *21*(sup1), 87-100. doi: 10.1080/12265934.2016.1236700

Zhang, Y., Qiu, M., Tsai, C.-W., Hassan, M. M., & Alamri, A. (2017). Health-CPS: Healthcare Cyber-Physical System Assisted by Cloud and Big Data. *IEEE Systems Journal*, *11*(1), 88-95. doi: 10.1109/JSYST.2015.2460747

Zhihui, L., Qian, C., Yonghua, Z., & Rui, Z. (2018). Signal Cooperative Control With Traffic Supply and Demand on a Single Intersection. *IEEE Access*, *6*, 54407-54416.

Zhou, Z., De Schutter, B., Lin, S., & Xi, Y. (2017). Two-Level Hierarchical Model-Based Predictive Control for Large-Scale Urban Traffic Networks. *IEEE Control Syst. Technol.*, *25*(2), 496–508. doi: 10.1109/TCST.2016.2572169