2019-04-16

Hybrid DES-based Vehicular Network Simulator with Multichannel Operations

Le Wang
Worcester Polytechnic Institute

Hybrid DES-based Vehicular Network Simulator with Multichannel Operations

Le Wang

A Dissertation
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy
in
Electrical and Computer Engineering

April 2019

APPROVED:

Professor Alexander M. Wyglinski
Primary Advisor
Worcester Polytechnic Institute

Professor Andrew Clark
Committee Member
Worcester Polytechnic Institute

Professor Kaushik Chowdhury
Committee Member
Northeastern University

Abstract

Vehicular Ad-hoc Network (VANET) is considered to be a viable technology for inter-vehicle communications for the purpose of improving road safety and efficiency. The Enhanced Distribution Channel Access (EDCA) mechanism and multichannel operations are introduced to ensure the Quality of Service (QoS). Therefore, it is necessary to create an accurate vehicular network simulator that guarantees the vehicular communications will work as described in the protocols.

A comprehensive vehicular network simulator should consider the interaction between mobility models and network protocols. In this dissertation, a novel vehicular network simulation environment, *VANET Toolbox*, designed using discrete-event system (DES) is presented. The APP layer DES Module of the proposed simulator integrates vehicular mobility operations with message generation functions. The MAC layer DES module supports single channel and multichannel EDCA operations. The PHY layer DES module supports bit-level processing. Compared with packet-based simulator such as NS-3, the proposed PHY layer is more realistic and accurate.

The EDCA scheme is evaluated and compared with the traditional Carrier-Sensing Multiple Access (CSMA) scheme, with the simulations proving that data with different priorities can coexist in the same channel. The multichannel operation for the EDCA scheme is also analyzed in this dissertation. The multichannel switching operation and coordination may cause packet dropping or increased latency to the communication. The simulations show that with heavy network traffic, multichannel communication performs better than single channel communication. From the perspective of safety-related messages, the multichannel operation is able to isolate the interference from the non-safety messages in order to achieve a better packet delivery rate and latency. On the other hand, the non-safety messages can achieve high throughput with reasonable latency from multichannel communication under heavy load traffic scenario.

Acknowledgements

This research is supported by generous contributions of the MathWork Inc. I want to thank Wei Li from MathWorks, who has spent tremendous of time teaching me on MATLAB DES voluntarily. I would also like to thank my committee members, Professor Clark and Professor Chowdhury. Their comments and guidance of my work help me become a better researcher.

Personally, I would specifically like to thank my advisor, Professor Alexander Wyglinski, who provides me this opportunity to do academic research on vehicular network area. I would never succeed without his excellent guidance and support. I will be forever grateful to him.

Over the past few year, Wireless Innovation Lab is an enthusiastic place to share research ideas. Supports from my colleagues, Renato, Paulo, Travis, Kuldeep, Kyle, Nivetha is always something I could rely on. Specifically, I want to thank Renato Iida, who designed the wireless channel of VANET Toolbox and provides huge help on debugging the simulator.

As always, my parents have supported me during the course of this degree and throughout my life. I am deeply indebted to them for their unconditional love and encouragement.

I also want to express my appreciation to my best friend, Zoe, for her kindly generous encouragement during my eight years of life in U.S..

Finally, I am fortunate enough to meet my wife Yuqing Yang, who always supports me during my graduate studies. This is a stepping stone for our life together and was only possible because of her.

Contents

List of Figures

List of Tables

List of Abbreviations

AC	Access Categories
ACK	Acknowledgment
AIFS	Arbitration Inter-Frame Spacing
APP	Application
AWGN	Additive White Gaussian Noise
BPSK	Binary Phase Shift Keying
BS	Base Station
BSS	Basic Service Set
BSM	Basic Safety Message
BSW	Blind Spot Warning
CDF	Cumulative Distribution Function
CI	Confidence Interval
CFP	Contention Free Period
CFM	Car following model
CG	Code Generation
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sensing Media Access / Collision Avoidance
CT	Channel Tracking
CPP	Cluster Point Process
CW	Contention Window
CWmin/CWmax	Minimum CW / Maximum CW
DCF	Distributed Coordination Function

DES	Discrete-Event System
DIFS	DCF Inter-Frame Spacing
DSRC	Dedicated Short Range Communication
EDCA	Enhanced Distributed Channel Access
FIFO	First In First Out
FFT	Fast Fourier Transform
GPS	Global Positioning Systems
GUI	Graphical User Interface
HCCA	HCF Controlled Channel Access
HCF	Hybrid Coordination Function
IFS	Inter-Frame Spacing
IoT	Internet of Things
ISI	Inter-Symbol Interference
ITS	Intelligent Transportation System
LCM	Lane-Changing Mechanism
LIFO	Last In First Out
L-LTF	Legacy-Long Training Field
LOS	Line of Sight
L-SIG	Legacy Signal field
L-STF	Legacy Short Training Field
LTE	Long-Term Evolution
LTF	Long Training Field
LTI	Linear Time Invariant
MAC	Media Access Control
MT	Multi-threaded
Non-HT	Non-High-throughput
OBU	OnBoard Unit
OFDM	Orthogonal Frequency Division Multiplexing
OOP	Object Oriented Programming

OS	Operating System
PCF	Point Coordination Function
PDF	Probability Distribution Function
PDR	Packet Delivery Rate
PER	Packet Error Rate
PHY	Physical
PLCP	Physical Layer Convergence Protocol
PSDU	PLCP Service Data Unit
PSR	Packet Success Rate
QoS	Quality-of-Service
RDT	Reliable Data Transmission
RMSE	Root Mean Squared Error
RSU	RoadSide Unit
RSSI	Received Signal Strength Indicator
Rx	Reception
SAE	Society of Automotive Engineering
SDR	Software-Defined Radio
SG	Stochastic Geometry
SIFS	Short Inter-Frame Spacing
SINR	Signal To Interference Plus Noise Ratio
SN	Sequence Number
ST	Single Threaded
STF	Short Training Field
Tx	Transmission
WAVE	Wireless Access in Vehicular Environments
WBSS	WAVE Basic Service Set
WSM	WAVE Short Message
WSMP	WAVE Short Message Protocol
UP	User Priority

V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-everything
VANET	Vehicular Ad-hoc Network

Chapter 1

Introduction

1.1 Motivation

In 2016, National Highway Traffic Safety Administration (NHTSA) announced that 37,461 people were killed in 34,436 vehicle crashes, an average of 102 per day [1]. Self-driving technology is one strategy that promises to save lives. The automotive sector has been focused on technologies that enable self-driving vehicles for years [2–6]. Currently, the self-driving technologies are focusing on the auxiliary sensors such as the Light Detection and Ranging (LiDAR) technology [7] and vision cameras [8, 9]. Unfortunately, most sensors are facing restrictions. For instance, the commercially available LIDAR devices are as expensive as $75,000 USD, which is even higher than the cost of a general vehicle itself [10]. The vision camera technology usually needs a line of sight environment. On the contrary, the wireless communication technology is not strictly constrained by the transmission environment and, due to its low cost, it has already been widely adopted by varieties of portable devices such as phones [11] and laptops.

Over the past several years, wireless communication and networks have become important across several domains, including the automotive sector. Ever since the first generation analog communication system in 1970 to the high speed digital networks we use everyday, wireless networks have been improving dramatically. Consequently, the automotive sector has started to leverage wireless communication networks in order to increase the situational

awareness level of self-driving cars as well as the reliability of the information they gather from the surrounding environment. These vehicles using wireless communication technology enables them to talk to each other with shared safety and mobility information; these vehicles are called *connected vehicles*. Connectivity promises more situational awareness, thus achieving the ability of these vehicles to save lives [12].

Vehicular networking has been extensively researched over the past two decades [13]. Given the complex nature of the operating environment, including the rapidly changing network topology [14], time-varying physical characteristics of the propagation medium [15, 16], and the need for a robust medium access control (MAC) protocol [17], vehicular networking is a challenging research area being addressed by both academia and industry.

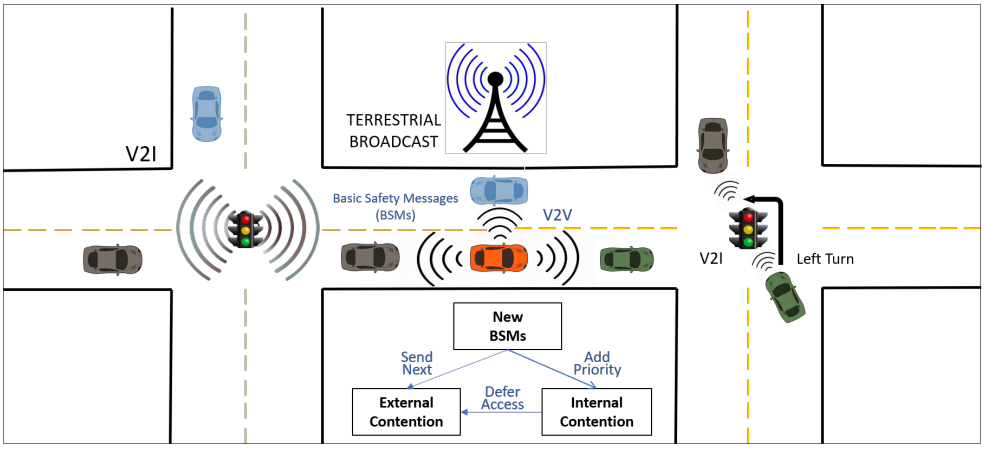


Figure 1.1: Interoperability on VANET/DSRC: V2I and V2V. The figure shows examples of BSMs via V2V and *left turn signal* via V2I. A simplified Finite State Machine (FSM) of EDCA in VANET/DSRC is also shown.

Vehicular Ad-hoc Networks (VANETs) are one type of Mobile Ad-hoc Networks (MANETs) that are specifically designed for moving cars. A VANET supports two types of communications: vehicle-to-vehicle (V2V) [18] and vehicle-to-infrastructure (V2I) communications [19], as shown in Figure 1.1. For V2V communications, the vehicles are equipped with an On Board Unit (OBU). The V2V communication is deployed between the vehicular OBUs for the purpose of road safety and traffic management applications [20]. Measurements for V2V DSRC are presented in [18]. For V2I communications, a RoadSide Unit (RSU) is installed to traffic infrastructure, such as traffic lights. The V2I communication is performed between

the RSUs and the OBUs [21, 22].

One of the applications provided by VANET is road safety, which is one of the primary motivations for implementing vehicular communications. The National Highway Traffic Safety Administration (NHTSA) defined several scenarios of DSRC in DOT-HS-821-014 [23]. The maximum delay for the vehicles to receive these emergency messages affects their ability to handle actions in order to avoid accidents. Reference [24] states that vehicular networks have the potential to save time and save lives. Therefore, vehicular network applications can be classified into two types: safety-related applications, and efficiency applications (non-safety applications).

In 2011, Vehicle Safety Communications Applications (VSC-A) organization summarized eight crash scenarios based on the statistics of vehicle accidents obtained from the US government with respect to frequency, cost, and casualties [25]. To address these crash scenarios, VSC-A proposed several safety applications such as Emergency Electronic Brake Lights (EEBL) [26], Blind Spot Warning (BSW)/Lane Changing Warning (LCW) [27], and Intersection Movement Assist (IMA) [28]. These safety applications are often evaluated in the vehicular mobility models [29]. In order to guarantee the safety applications will work as described in the standards, it is necessary to create accurate vehicular network simulations.

1.2 State-of-the-Art

Vehicular network simulation can be classified into hardware simulations and software simulations. One example of hardware simulation is the Mcity project of University of Michigan, which owns a 32-acre mock city and 1,500 vehicles [30]. The cost of Mcity in 2015 is 10 million dollars with an extra 15 million dollars invested in 2018 [31]. On the other hand software simulations are another option to provide an evaluation environment of vehicular network performance with low cost and decent accuracy.

A software vehicular network simulator is usually complicated because it has to consider vehicular mobility and communication network simultaneously. The position and speed of vehicles may impact the quality of wireless communications, and the shared information over a vehicular communication network could influence the vehicular path and mobility

decisions. This strong interaction requires the traffic mobility simulators to work closely with vehicular network simulators [32].

Unfortunately, compatibility issues have become a significant challenge because both types of simulators have been designed to be controlled separately. This interaction has almost never been considered as the primary goal when designing simulators. Over the past few years, vehicular network researchers have worked hard to create an interface between the two simulation fields and several achievements have been proposed. Based on the level of interactions, we classify the simulators as *joint simulators* and *integrated simulators*.

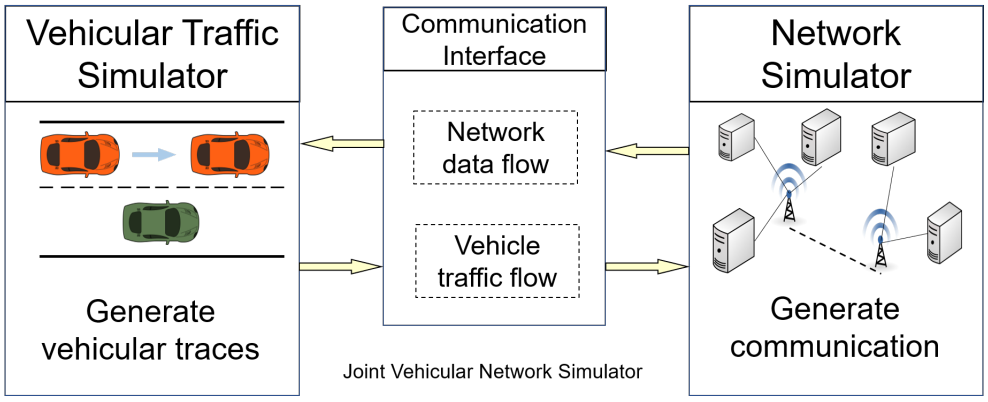


Figure 1.2: Interaction in Joint Network and Traffic Simulator. Vehicular traffic flows are created by vehicular traffic simulator and passed to the network simulator via the 3rd party communication interface. The network simulator then performs network simulation based on the traces information and returns the results through the communication interface. The vehicular traffic simulator alters the vehicles' movement according to the network communication result.

For the *joint simulators* shown in Figure 1.2, an interface is created to associate the existing traffic mobility simulators and network simulators. An example is the iTetris [33] project, which associates the traffic simulator SUMO [34, 35] with the network simulator NS-2 [36, 37], NS-3 [38, 39], or OMNET [40]. Another example is using TraCI [41–43] to connect SUMO to other simulator like OMNet++ or MATLAB. The interface here plays a role of relaying messages between the simulators. Traffic flows are extracted from SUMO and sent to the network simulator through the interface, and instructions from the network simulators are sent to SUMO to alter the traffic. The advantage of this cross-layer joint approach is to enjoy the benefits of both well-developed simulators. However, one limitation

is the design complexity of the interface as it needs to let both simulators run simultaneously. Another limitation is the configuration complexity, as the users usually need to tweak a large number of parameters on both simulators to make the simulation work correctly.

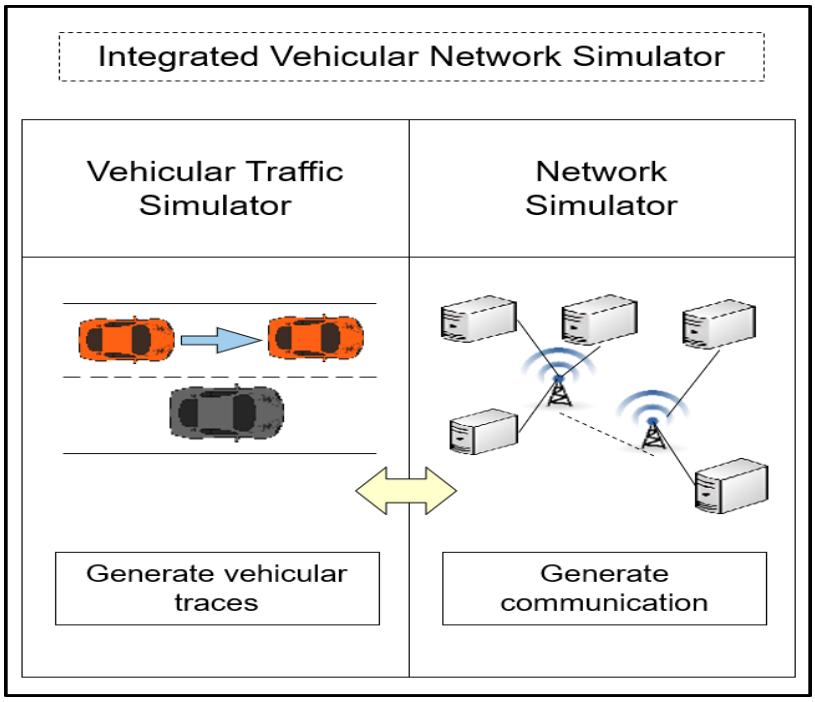


Figure 1.3: Interaction in Integrated Network and Traffic Simulator. The vehicular trace information is passed to the network simulator directly, where the latter performs network simulation and returns the results. Then the vehicular traffic simulator alters the vehicles' movements in real time.

Another solution is to combine network and traffic simulators into one single simulator for the purpose of full interaction. This type of simulator is referred to as an *integrated simulator* as shown in Figure 1.3, which has the capability to have both simulators work and interact flawlessly. Several examples are MoVes [44], NCTUns simulator [45] and, VISSIM [46]. The limitations mainly come from the oversimplified network or mobility models. For example, several simulators only have a basic radio propagation model with CSMA/CA as the MAC layer [47].

The PHY layer model in NS-3 integrates several VANET characteristic components at the packet level. One difference between MATLAB and NS-3 platform in a VANET simulation is how the PHY layer is implemented. The proposed MATLAB-based VANET

simulator in this dissertation performs all the signal processing on the bit level and transmits wireless signals using the MATLAB WLAN System Toolbox as in [48], which increases accuracy when compared to NS3 with respect to model bit error rates at the receiver end [49]. Furthermore, in a MATLAB/Simulink model, it is feasible to replace the simulated wireless channel with real radio hardware such as USRP [50, 51].

1.3 Research Contributions

In this dissertation, we present an integrated vehicular network simulator, *VANET Toolbox*, works in MATLAB/Simulink environment, as shown in Figure 1.4. *VANET Toolbox* covers the main stack of vehicular network protocols including the application (APP) layer, the medium access control (MAC) layer, and the physical (PHY) layer. Several mobility models including car following, lane changing, as well as intersection management are embedded in the APP layer. The design purpose of *VANET Toolbox* is to provide a framework of a vehicular simulation environment so that more developers can continue to improve it.

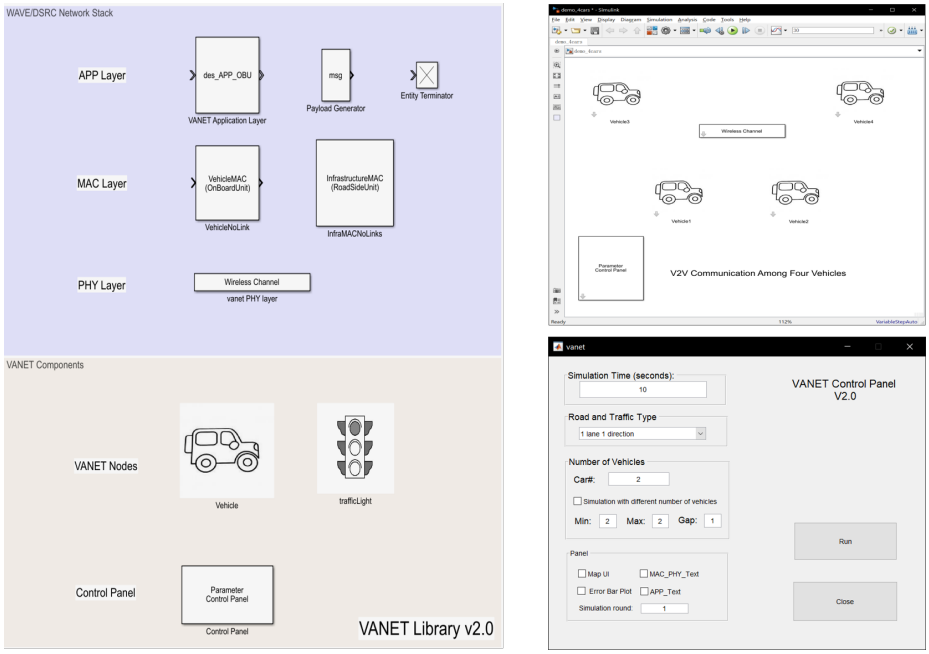


Figure 1.4: Overview of VANET Toolbox Library and Simulation Panel. The left section is the VANET Library containing Simulink blocks of VANET Toolbox. The right section shows examples of running simulation via Simulink model or VANET UI Panel.

The contributions of this paper are summarized as followings:

- The proposed novel vehicular network simulation environment is an integrated type simulator combining both vehicle traffic simulation and network simulation together. It supports a hybrid of time-driven and event-driven simulation environment. The design framework of multichannel operation is provided in details and it can be applied to any DES programming language.

- The performance of single channel communication is evaluated with the proposed simulation environment. The proposed bit-level PHY layer shows its accuracy when compared with packet-level NS-3. The simulations prove that the EDCA scheme in the single channel scenario is able to allow data with different priorities to coexist in the same channel and the EDCA scheme is effective in ensuring the quality of service (QoS) of safety-related messages with higher priority. Two coordinated lane changing schemes based on single channel V2V communication are evaluated and the overall traffic efficiency can be enhanced by the vehicular communication.

- The performance of a multichannel EDCA mechanism is evaluated with all four ACs traffic levels enabled, *i.e.,* the data flows within $4ACs \times 7channels = 28ACs$. The results prove that the multichannel scheme will bring a high packet delivery ratio and a high throughput to the overall network traffic, while at the same time the channel switching operation may cause a obvious impact on safety-related services with strict latency requirements. Additionally, a cooperative SCH/AC reservation scheme is proposed, which could adaptively decide the SCH frequency as well as AC priority according to the SCH/AC information received from nodes with at most two-hop distance. The simulation result shows the proposed scheme is able to increase the channel utilization with multiple priorities, thereby reducing the contention probability to achieve an improvement on packet delivery latency.

1.4 Dissertation Outline

The rest of this dissertation proposal is organized as follows: Chapter 2 discusses the theoretical fundamentals used in this work. This chapter starts from the classification of a system, then introduces the concept of a general Discrete-Event System (DES). A DES can be described by *Automata* or *State Transition Diagram*. A DES simulator is usually designed based on timed automata or event scheduling scheme. Next, this chapter explains the three basic elements in a MATLAB DES and shows how to create a simple MATLAB DES step-by-step. Finally, the theory and protocols of a vehicular network are introduced.

In Chapter 3, a prototype vehicular network simulator, *VANET Toolbox*, is devised and implemented. The chapter focuses on the structure of APP, MAC and PHY layers and the implementations of these layers using MATLAB DES. Then, other simulator related features are also implemented. These features are necessary to a vehicular network simulator and implemented using MATLAB/Simulink. Finally, the methods to create a model using *VANET Toolbox* are shown and the advantages and limitations of each methods are discussed.

In Chapter 4, the performance of *VANET Toolbox* is evaluated. The analysis starts from the computational costs in terms of events of a general vehicular network model. Then, a case study of V2V communication is performed. We first show how the EDCA protocol ensures the quality-of-service (QoS) of messages with higher priority under heavy network traffic. Next, two lane-changing schemes based on V2V communications are proposed. The limitations of *VANET Toolbox* are provided at the end of the chapter.

Chapter 5 describes the multichannel operations including channel switching operation and coordination. The design framework of multichannel operation in DES is provided and the its implementation in MATLAB/Simulink is also provided. A series of simulations on both single channel and multichannel communication are conducted. The results shown that multichannel operations are able to guarantee the QoS of safety-related services with respect to a lower latency and a higher packet delivery ratio meanwhile providing high throughput to non-safety services.

1.5 Resulting Peer-Reviewed Publications and Software Projects

Throughput this dissertation, I have published my research work in high-impact, peer-reviewed venues.

Peer-Reviewed Journal Papers

- L. Wang and A.M. Wyglinski, "Multichannel EDCA Operations for Vehicular Networks via Discrete Event System", IEEE Access, submitted, 2019. Impact factor: 3.557 (2017).
- L. Wang, R. Iida and A. M. Wyglinski, "Vehicular Networking Simulation Environment via Discrete Event System Modeling," IEEE Access, under 2nd review, 2019. Impact factor: 3.557 (2017).

Peer-Reviewed Conference Papers

- L. Wang, R. Iida and A. M. Wyglinski, "Performance Analysis of Multichannel EDCA-based V2V Communications via Discrete Event System", submitted, in Vehicular Technology Conference (VTC Fall), 2019 IEEE 90th, Sept 2019.
- L. Wang, R. Iida and A. M. Wyglinski, "Coordinated Lane Changing Using V2V Communications," in Vehicular Technology Conference (VTC Fall), 2018 IEEE 88th, Aug 2018.
- L. Wang, R. Iida and A. M. Wyglinski, "Performance Analysis of EDCA for IEEE 802.11p/DSRC based V2V Communication in Discrete Event System," in Vehicular Technology Conference (VTC Fall), 2017 IEEE 86th, Sept 2017.

Software Project: VANET Toolbox

The vehicular network simulator, *VANET Toolbox*, is open source and can be downloaded from the File Exchange MATLAB Central by searching 'VANET Toolbox'. A Github link for the VANET Toolbox repository is also provided in [52].

Chapter 2

Fundamentals of DES Simulation and Vehicular Network

This chapter introduces all necessary knowledge to build a MATLAB discrete event system (DES). The chapter starts from the definition of a system. From the classification of systems, DES is defined as a discrete-state and event-driven system. A DES can be simulated by *timed automata or event scheduling scheme*. Next, the basic elements (entity, event,action) of a MATLAB DES are introduced and a sample MATLAB DES is created step by step. The last section of this chapter discusses DSRC/WAVE standards, which are the theoretical basis of *VANET Toolbox*.

2.1 Discrete Event Systems

A Discrete event system (DES) is a special class of *systems*, which is an abstract model of a real discrete-state event-driven system to be simulated [53]. The abstract system may include numerous *system states* that occur at discrete instants of time. The transitions of system states are described as *Events*. The idea of *event-driven* is similar to the *interrupt* feature in computer systems. A computer system is designed to cope with asynchronous events that may occur at any time. These events are independent to the computer clock. Therefore, the technology, especially with computers involved, is usually an event-driven

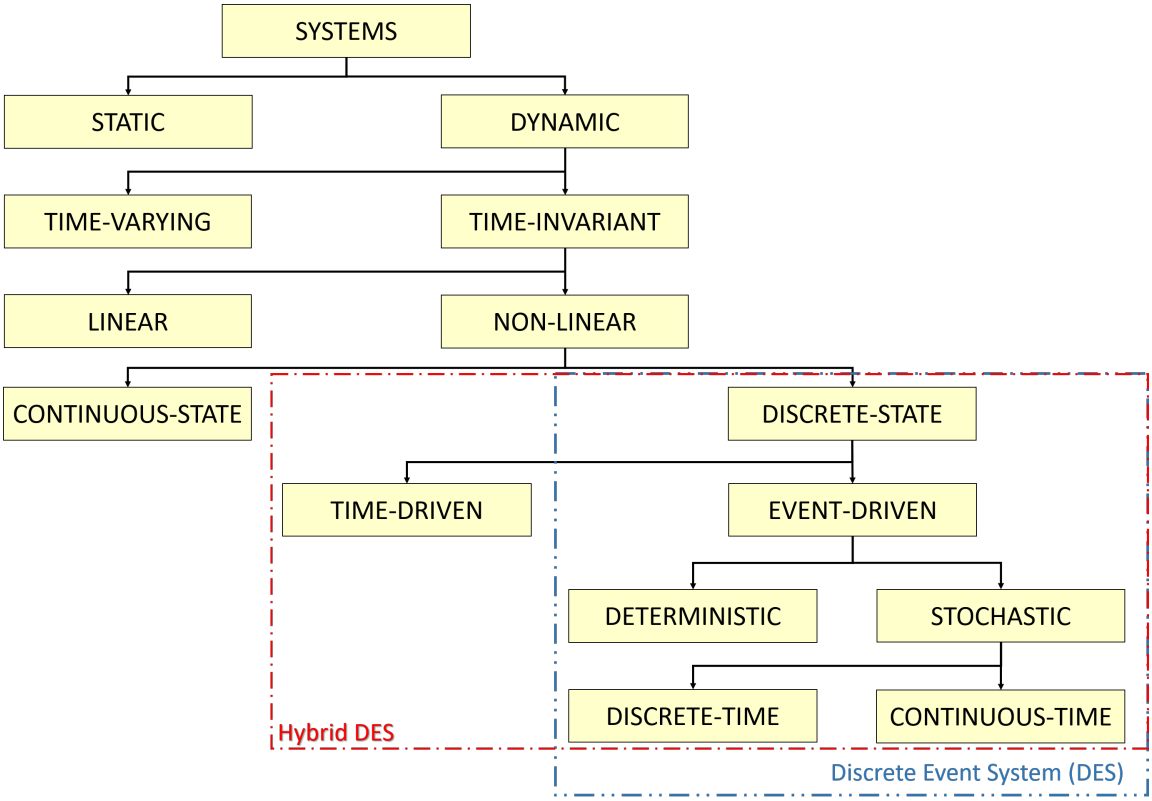


Figure 2.1: Systems overview and Discrete Event Systems. Discrete Event System is classified as a discrete-state event-trigger system. A hybrid system including both time-driven and event-driven DES is suitable for network PHY layer simulation.

system such as communication systems with message timeout features.

2.1.1 Discrete Event System Concept

When a system can be described as a set of discrete states, and the state transitions are caused by events which occur instantaneously, this system is a Discrete Event System (DES), *i.e.,* a DES is a *discrete-state*, *event-driven* system, and its state transitions are entirely caused by the asynchronous discrete events over time as shown in Figure 2.1. In a DES, 'time' is no longer the key factor to drive the system. The set of *events* replace the role of *time* and serves the purpose of driving a DES, with each 'event' causing a state transition. A DES may be modeled in either continuous time or in discrete time.

Even though a DES is defined as a discrete-state, event-driven dynamic system, it is

worth noting that a hybrid DES is more general when both time-driven and event-driven are present, as shown in Figure 2.1. For example, the operating system (OS) in a computer is designed to not only respond to asynchronous events occurring at any time but also processing functions synchronized by the computer clock. A hybrid DES may be deterministic or stochastic, and it can be modeled in either discrete or continuous time. More details about the classification of systems are provided in Appendix A.

The event set E is thought of as an alphabet of a DES. A sequence of events taken from the event set E becomes a string. A *language* defined over an event set E is a set of finite-length strings formed from events taken from E. Suppose an event set is $E = \{a, b, g\}$, a example of language L can be $L = \{\epsilon, a, abb\}$, in which ϵ is empty string, a is a single event and abb is a string of events taken from E with duplicate events.

As mentioned in the previous sections, a continuous state system can be modeled either by differential equations for continuous-time system or difference equations for a discrete-time system. Modeling a discrete event system is more complicated. In this section, a basic modeling mechanism, named *Automata*, for discrete event system is introduced. The concept of an automaton is shown through the *state transition diagram* (see Figure 2.2).

Suppose an event set is $E = \{a, b, g\}$. The circles in Figure 2.2 represent *states* and the arrows indicate the *transitions* between states. The *state space* of the automaton is $X = \{x, y, z\}$. The tags above the state transition arrows are event elements from the event set E. The transition function of the automaton is denoted as $f : X \times E \to X$. For example, $f(x, g) = y$ means the automaton is in state x. After event g happens, the automaton transits the state to y, *i.e.,* $x \to y$. The trigger of event a may be either an external input to the system, or an event generated by the system itself. The initial state is denoted by x_0, and a subset $X_m \subseteq X$ named *marked states* is identified as double circles. Marked states indicate states that have special meanings.

Table 2.1: All transition functions showing in Figure 2.2.

$f(x, a) = x$	$f(x, g) = y$
$f(z, g) = z$	$f(z, b) = x$
$f(y, b) = b$	$f(y, a) = f(y, g) = z$

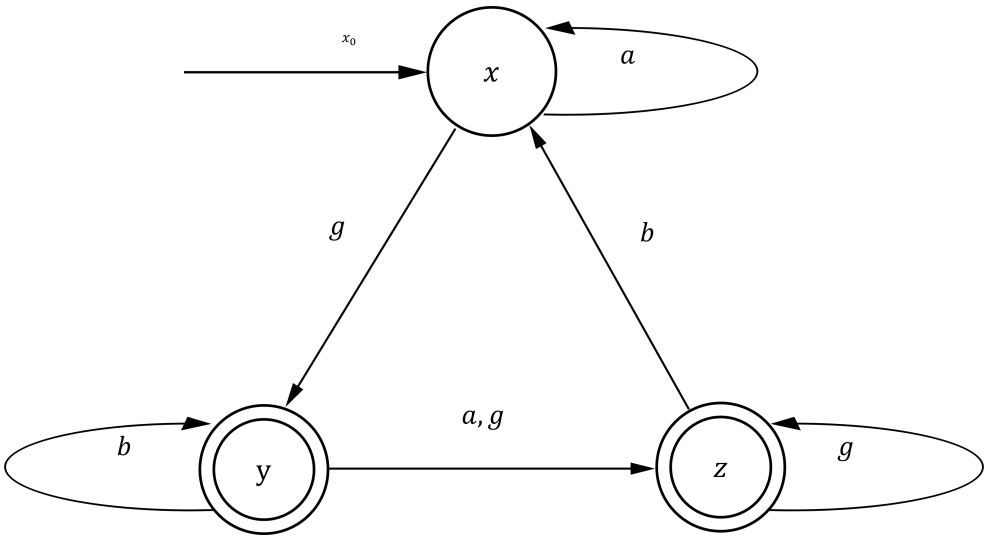


Figure 2.2: A simple automaton with state transition. States x, y, z may transit to another state via state transition functions caused by events.

All transition functions shown in the automaton of Figure 2.2 are listed in Table 2.1. From the transition functions, we observe the state may not change even after an event is occurred, as in $f(x, a) = x$, where the state remains in x after event a happened. Furthermore, different events triggered on the same states may cause the same state transition, such as $f(y, a) = f(y, g) = z$. In this case, both event a and event g can transit the initial state y to state z. Thus, it is difficult to distinguish between event a and event g by only observing the state transition $y \to z$.

A *deterministic automaton* is defined in Eq. (2.1). The annotations in the deterministic automaton are summarized in Table 2.2. A deterministic automaton indicates the state transition function f is derived from $X \times E \to X$, that is, only one state transition occurs within the same events. In contrast, for the *nondeterministic* automaton, the state transition function f is defined by $X \times E \to 2^X$. In this case, more than one state transitions may occur with the same event.

$$G = (X, E, f, \Gamma, x_0, X_m). \tag{2.1}$$

The automaton G starts in the initial state x_0 and after an event $e \in \Gamma(x_0) \subseteq E$ has occurred, it transits the state from x_0 to $f(x_0, e) \in X$. Then, the process continues based

Table 2.2: Notations and Definitions of a DES automaton .

Notations:	Definitions:	Instances:
E	discrete event set	$E = \{a, b, g\}$
e	an element event of E	$e = a$
ϵ	a string contains no events	$\epsilon = emptystring$
$\|s\|$	length of string s	$\|\epsilon\|== 0$
X	state space	$X = \{x, y, z\}$
X_m	set of marked states	$X_m \subseteq X$
x_0	initial state	
f	transition function	$f(z, b) = x$
$\Gamma(x)$	feasible event function of automaton at x	$\Gamma(x) \rightarrow f(x, e)$

on which state transition function f is inputed or generated to the system. Some examples of state transition functions are listed as:

$$f(x, \epsilon) = x$$

$$f(y, bab) = f(f(y, ba), b) = f(f(f(y, b), a), b) = f(f(y, a), b) = f(z, b) = x$$

2.1.2 Discrete Event Simulation

The above section has described how the state of a DES evolved as a result of event occurrences over time. The goal of this section is to use a model in order to obtain explicit mathematical expressions for quantities of interest. Simulation is a process through which a system model is evaluated numerically, and the data from this process are used to estimate various quantities of interest [53]. Several discrete-event simulation models are introduced in this section.

Timed Automata

The automaton $G = (X, E, f, \Gamma, x_0)$ we mentioned previously is referred to *untimed automaton* as *time* is not involved. In order to define a timed model of a DES, a *Clock*

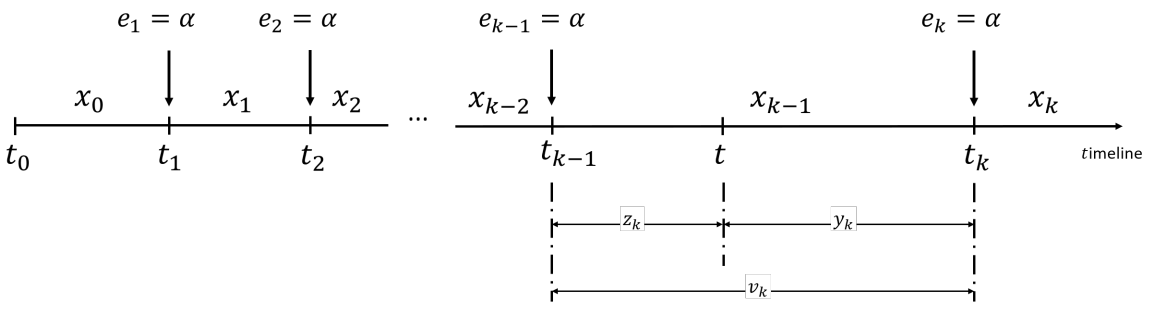


Figure 2.3: Clock Structure of a DES with Single Event $E = \{\alpha\}$.

Structure is introduced. We start from the simplest DES, that is, a single event $E = \{\alpha\}$ in a DES shown in Figure 2.3. The feasible event set is $\Gamma(x) = \{\alpha\}$ for all $x \in X$. The notations of the automaton $G = (X, E, f, \Gamma, x_0)$ are recorded in Table 2.2. The event sequence in this DES is $\vec{e} = \{e_1, e_2, ..., e_k\}$ and $e_1 = e_2 = ... = e_k = \alpha$. *Event Lifetime* denoted by v_k is defined as the length of the time interval of two successive events. For the single event DES, the kth lifetime of the event is defined as:

$$v_k = t_k - t_{k-1}, \quad k = 1, 2, ..., k, \quad v_k \in \mathbb{R}^+. \tag{2.2}$$

At time t_{k-1}, the kth event, e_k, is *enabled* with a lifetime v_k. A timer attached to e_k starts to count down from v_k. At time $t_k = t_{k_1} + v_k$, the timer reaches 0, e_k has to occur, a state transition is caused from x_{k-1} to x_k. Then, the same process repeats with the $(k+1)$th event, e_{k+1}.

An event has three status values: *active*, *enabled*, and *occurring*. The event α is *active* as long as it is feasible in the the current state, *i.e.*, $\alpha \in \Gamma(x)$. The event e_k is *enabled* when the last event, e_{k-1}, occurs and a state transition is caused. An event is *occurring* when its timer ticks down to 0, and a state transition takes place.

Suppose t is any time instant $t_{k-1} \leq t \leq t_{k+1}$ as shown in Figure 2.3. Then, t divides the time interval $[t_{k-1}, t_k]$ into two sections, named *clock* of the kth event:

$$y_k = t_k - t, \tag{2.3}$$

and *age* of the kth event is:

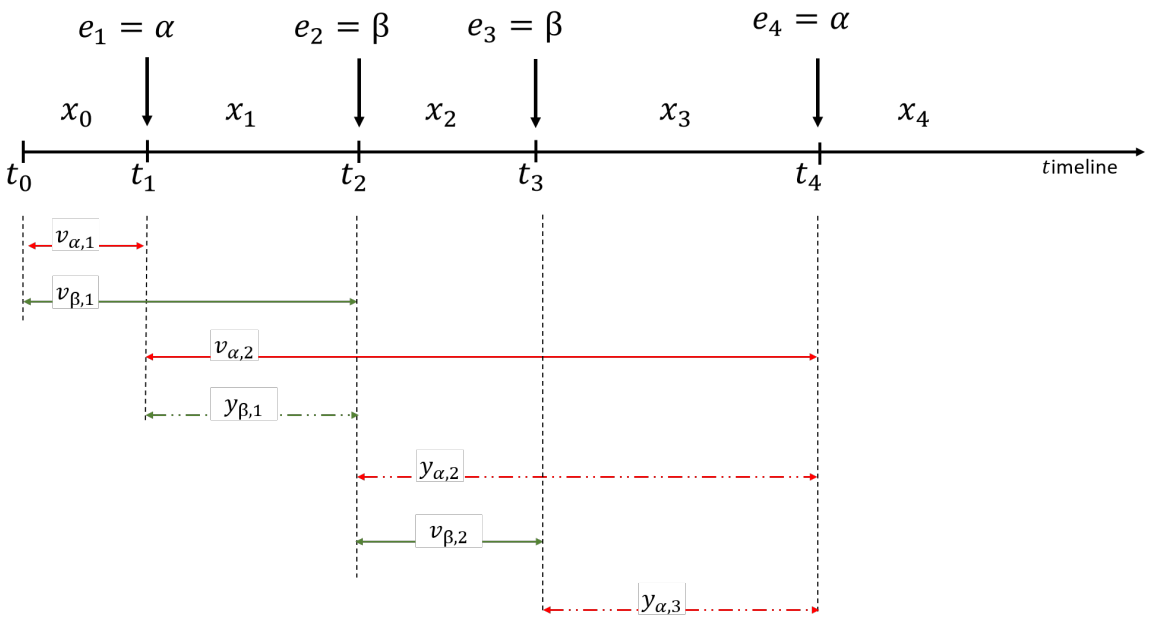


Figure 2.4: Clock Structure of a DES with double Events $E = \{\alpha, \beta\}$.

$$z_k = t - t_{k-1}. \tag{2.4}$$

Subsequently, we get the expression:

$$v_k = z_k + y_k. \tag{2.5}$$

Based on the analysis above, a DES can be specified by the *clock sequence* of events, that is, $\vec{v} = \{v_1, v_2, ..., v_k\}$.

A DES with more than one event is more complicated. Suppose a DES has two events, $E = \{\alpha, \beta\}$, and both events are always feasible, *i.e.*, $\Gamma(x) = \{\alpha, \beta\}$ for all $x \in X$. Then, two clock sequences of lifetimes for both events are specified by:

$$\vec{v}_\alpha = \{v_{\alpha,1}, v_{\alpha,2}, ..., v_{\alpha,k}\}, \quad \vec{v}_\beta = \{v_{\beta,1}, v_{\beta,2}, ..., v_{\beta,k}\}, . \tag{2.6}$$

Starting from time t_0 in Figure 2.4, both event α and event β are enabled, thus which event will occur first becomes a question. The winner is selected by comparing $v_{\alpha,1}$ with

$v_{\beta,1}$ and selecting the event with the shortest lifetime. Suppose $v_{\alpha,1} < v_{\beta,1}$, then event α will be the first event to occur at time $t_1 = t_0 + v_{\alpha,1}$.

Once event α has occurred, the state transition $x_0 \to x_1$ is caused and the DES needs to figure out which event occurs next. Since event β is still enabled at time t_1, the clock value of β is $y_{\beta,1} = v_{\beta,1} - v_{\alpha,1}$. At the same time, a new α is enabled. Since the old α just took place, the new α is defined with a new lifetime $v_{\alpha,2}$, selecting from the given clock sequence Eq. (2.6). If $v_{\beta,1} < v_{\alpha,2}$, β has a smaller clock value than α, then β is selected to occur at time $t_2 = t_1 + y_{\beta,1}$.

To summarize, the mechanism of choosing the *next event* for a DES with more than one events is by comparing *clock values* and picking the event with the smallest clock value. Whenever an event has occurred, its clock is reset to a new lifetime value from its clock sequence.

The event scheduling scheme

Simulation is a systematic means for generating sample paths of a DES, as shown in Figure 2.3 and 2.4. The event scheduling scheme is one approach to generate a sample path. Recall that E is a countable *event set*, and X is a countable *state space*. The initial state $x_0 \in X$ is given at time $t_0 = 0$ and the feasible events set is $\Gamma(x) \subseteq i$. Each state has a feasible events set $\Gamma(x)$, the events from $\Gamma(x)$ are the only events which may occur at this state.

For the initial event x_0, its feasible events set is $\Gamma(x_0)$. Suppose a feasible event $i \in \Gamma(x_0)$, has a clock value is attached, which indicates the period required before event i occurs. The clock value for an event i is y_i and its lifetime is v_i. For all $i \in \Gamma(x_0)$, we have $y_i = v_i$.

More generally, a state x, including the initial state x_0, has clock values y_i where $i \in \Gamma(x)$. The *triggering event* $e^{'}$ is the next event which will occur at that state x, *i.e.,* the event chosen with the smallest clock value (see Eq. (2.7)). When event $e^{'}$ occurs at state x, a new state $x^{'}$ is generated from the state transition function $f(x, e^{'})$.

$$e^{'} = \arg \min_{i \in \Gamma(x)} \{y_i\}. \tag{2.7}$$

The *inter-event time* , y^*, represents the amount of time spent at state x and is calculated by:

$$y^* = \min_{i \in \Gamma(x)} \{y_i\}. \tag{2.8}$$

The updated time is obtained by $t^{'} = t + y^*$. Once the new state $x^{'}$ is generated, the clock values for all feasible events are updated. If an event $i \in \Gamma(x^{'})$ and $i \neq e^{'}$ remains feasible in the new state $x^{'}$, then recall event β in Figure 2.4, where the new clock value is $y_i^{'} = y_i - y^*$. For all events which are not feasible in x but become feasible in $x^{'}$, *i.e.,* $e^{'} \in \Gamma(x^{'})$ but $e^{'} \notin \Gamma(x)$, a set of new lifetimes are supplied by the DES.

In the event scheduling scheme, whenever an event i is enabled at time t_n, its next occurrence is scheduled at time $t_n + v_i$, where v_i is a lifetime sample supplied by the DES. Thus, a Scheduled Event List (SEL) replaces maintaining the clock values $y_i, i \in \Gamma(x)$ where:

$$L = \{(e_k, t_k)\}, \quad k = 1, 2, ..., m_L. \tag{2.9}$$

given that m is the number of events in the events set E, m_L is the number of *feasible* events for the current state, *i.e.,* $m = |E|, m_L = |\Gamma(x)|, m_L \leq m$. The SEL is always ordered on a *smallest-scheduled-time-first* basis and shown in Figure 2.5.

The *Initialize* function sets the initial state to x_0 and the initial simulation time $t_0 = 0$. And it is also triggers the *Random Variate Generator* to create event lifetimes v_k for all feasible events $\Gamma(x)$. With a *Schedule Event List* (SEL) initialized, in which all entries are stored in increasing order of scheduled times, including e_1 being the triggering event and t_1 being the time when e_1 occurs, the DES simulation usually repeats the following six steps:

Step 1. Remove entry (e_1, t_1) from SEL.

Step 2. Update the simulation *Time* $t^{'} = t_1$.

Step 3. Update the *State* to new state $x^{'}$ according to $x^{'} = f(x, e_1)$.

Step 4. Delete all (e_k, t_k) from SEL such that $e_k \notin \Gamma(x^{'})$.

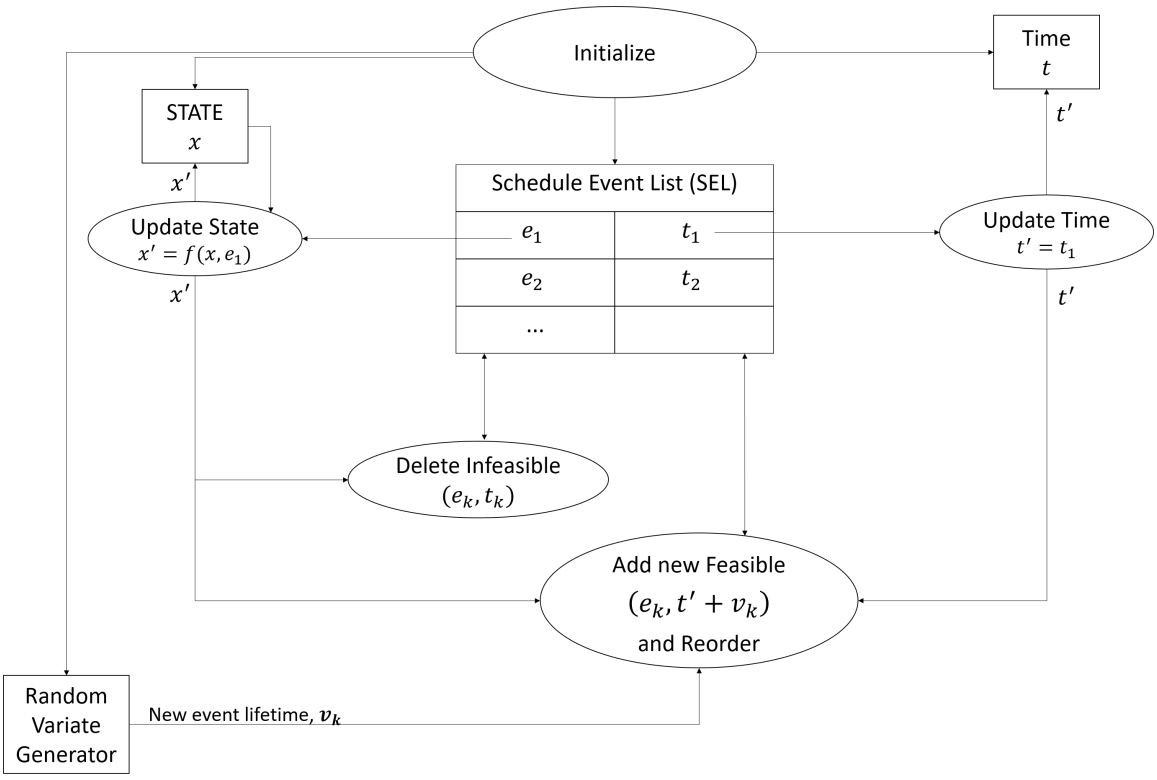


Figure 2.5: The Event Scheduling scheme in computer simulation.

Step 5. Add unscheduled feasible event(s) to SEL. For such event i, the event time is $Time + v_i$, in which *Time* is set from *Step 2* and v_i is obtained from the Random Variate Generator.

Step 6. Based on the *smallest-scheduled-time-first scheme* introduced in the last section, the SEL is reordered.

A DES simulator on computers based on the *Scheduled Event List* scheme has the following components:

1. *Initialization Routine:* Initializes all simulation data at the start stage of the simulation.

2. *Data Registers:* Stores data for estimation purposes.

3. *state list:* Stores all states variables.

4. *State Update routine*: Based on the next event, updates the state, *i.e.,* $x^{'} = f(x, e^{'})$.

5. *Time variable:* Stores the simulation time t.

6. *Time Update Routine*: Decides which is the next event to occur and advances the simulation time t to the occurrence time of that event.

7. *Scheduled Event List (SEL):* All scheduled events are stored in the list with their occurrence time.

8. *Random Variate Generation Routines:* Computer generates random numbers and are transformed into random variates based on the event lifetime distributions.

9. *Report Generation Routine:* Computes various interest based on the data collected during the simulation.

10. *Main Program:* Coordinates all components including *Initialization*, *State* and *Time*, then repeat the SEL steps introduced above. It is also responsible for stopping a simulation and generating the reposts.

The Process-oriented Simulation Scheme

In a DES, users or applications are often contending for service due to the limit of resources. These users or applications are considered as *'entities'*. The entities flow through the DES via a *process*, and a process is a sequence of events separated by time intervals. During a time interval, an entity is either under service or waiting for service. The process-oriented simulation scheme consists of several processes, one for each users or applications.

A DES simulator based on the *process-oriented simulation scheme* has the following components:

1. *Entities:* Objects requesting service, *e.g.,* data in a communication network. The *entity type* is characterized by process in the DES and a DES may contain more than

one type of entity types. For instance, in a communication network, data unit in network layer is *network entity* and in MAC layer is *frame entity*. Both entities follow different processes.

2. *Attributes:* A specific entity is attached with a unique attribute information containing this entity's attributes.

3. *Resources:* Objects providing service, *e.g.,* CPUs in a computer, routers in a communication network. Time delay mainly comes from service waiting time or service processing time.

4. *Queues:* A container of entities who are requesting and waiting for the same resource. An entity in a DES is either in the process of being served or waiting in some queue.

5. *Process:* A process is a sequence of functions including *Logic functions* and *Time delay functions*. A logic function is an instantaneous action such as checking if a resource is available or updating a data structure. The time delay function usually holds the entity for some period of time. There are two types of time delay functions:

   (a) *Specified time:* The delay is a fixed value. For instance, a service time in the simulation only depends on some preset service time distribution.

   (b) *Unspecified time:* The delay is not fixed depending on the state of the system. For instance, the entity waiting period before receiving the requested source.

The process-oriented simulation scheme is suitable for *queuing* systems. For example, the messages in the computer network are as entities flowing through a network of queues and servers. An event scheduling scheme is more general to model DES. A DES simulation language is usually based on the process-oriented simulation scheme, but also equipped with event scheduling scheme to model features that the process-oriented scheme cannot handle.

Discrete Event Simulation Tools

In order to simulate a specific DES of interest, one has two options: either build a DES simulator or use an existing DES simulator. Based on the event scheduling scheme and/or

process-oriented scheme, it is possible to create a DES model using standard computer languages such as C++. However, building a DES simulator is outside the scope of this proposal. As our goal is to simulate a DES of vehicular network, choosing some existing DES simulator is preferred. Several commercially available DES simulation software have been created, almost all of them are Object-Oriented Programming (OOP) with Graphical User Interface (GUI). Some of the DES software are introduced below:

- *General Purpose Simulation System* (GPSS) was developed by IBM and is current available through Minuteman Software with the most recent version, GPSS/H [54]. Using the building blocks (macros) provided by GPSS/H, users can create process-oriented simulation models in terms of block diagrams.

- *SIMulation ANalysis* (SIMAN) was developed by Pegden [55] and its latest version, 'Arena', is equipped with GUI [56]. Arena supports event scheduling scheme and process-oriented scheme with animation during the simulation. It is currently available through Rockwell automaton.

- *SIMSCRIPT* was developed by Markowitz at the Rand Corporation. The most recent version, SIMSCRIPT III [57], is available through the CACI Products Company. It supports both process-oriented and event scheduling scheme.

- *Simulation Language for Alternative Modeling* (SLAM) was developed by Pegden and Pritsker [58] and is available through the Pritsker Corporation. It supports both process-oriented and event scheduling schemes. SLAM provides GUI and animation for the modeling process.

- *EXTEND* is available through Imagine That, Inc. The feature of EXTEND is that users can create their own OOP objects to extend the libraries for different applications [59]. It supports hierarchical modeling and GUI as well as animation.

- *SimEvents* is a toolbox of MATLAB through The MathWorks, Inc [60]. SimEvents Toolbox was required to operate with Simulink, which is a time-driven simulator within MATLAB suite. Since MATLAB R2016a version, SimEvents is integrated with

a MATLAB Discrete Event System (DES) **system block**. The MATLAB DES system block allows users to create a event-driven system using MATLAB OOP languages. Therefore, the new SimEvents toolbox with MATLAB DES supports both time-driven and event-driven components in a hybrid way. More features are introduced in the next section.

There are other DES softwares such as GASP IV [61], SIMPAS [62], SIMULA [63], SIM++ [64] that are specifically target specific type of DES such as manufacturing systems and computer networks. In this proposal, the vehicular network simulator is developed by SimEvents with MATLAB DES, we will mainly focus on MATLAB DES and may introduce some SimEvents blocks when needed.

2.2 MATLAB Discrete Event System (DES)

MATLAB/Simulink is a numerical computing environment developed by the MathWorks. By using MATLAB languages, one can perform matrix calculations, plot functions and data, create user interfaces (UI) and cooperate with other programming languages such as C/C++, Java, Python. As of 2017, engineers and scientists on the order of million use MATLAB across industry and academia in engineering, science and economics [65].

SimEvents toolbox was working only in Simulink and providing a discrete-event simulation environment with blocks. Since R2016a, a new system object named *MATLAB.Discrete--EventSystem* joined the SimEvents family. This system object allows users to create an event-driven entity-flow system in object-oriented programming (OOP) MATLAB language, and use it in Simulink as a system block. The new SimEvents with MATLAB DES covers both MATLAB and Simulink platforms, which greatly extends the flexibility and scalability when modeling DES.

2.2.1 Features of MATLAB DES

A MATLAB DES has the following features:

- Simulink is a time-driven platform and MATLAB DES is an event-driven system. Therefore, a model developed by MATLAB DES supports both time-driven and

event-driven components. This feature makes MATLAB DES suitable for designing network/communication related simulation systems as the physical (PHY) layer of network is time related while the other layers are events-driven type.

- MATLAB DES allows users to take advantage of a rich collection of MATLAB tools such as WLAN System Toolbox, signal processing, and visualization. This feature makes MATLAB DES beyond a pure DES simulator, which makes it possible to develop a more comprehensive and complex system.

- The vehicular network simulator we developed using MATLAB DES is convenient to cooperate with hardware such as the USRP software defined radio. The expandability of MATLAB/Simulink saves a significant mount of time for developers as they do not need to test different tools during the developing process to support one specific feature. This *all-in-one* feature of MATLAB DES increases the development efficiency and decreases the chance of incompatible problems.

Since MATLAB R2017b version, MATLAB DES supports *code generation*. This feature allows users to convert MATLAB code into C++ code so that the simulaiton speed is greatly increased. A MATLAB DES has three elements: *Entity*, *Event* and *Action*.

Entity

The elements flow in a DES are called *SimEvents entities*. SimEvents entities contain static information and can be MATLAB built-in data type, structured/bus data types. A MATLAB DES may have one or more entity storages, with each storage containing SimEvents entities. These entities are sorted in certain order in the storages, such as a FIFO queue, a priority queue. A MATLAB DES can take entity as input or output, *i.e.,* a SimEvents entity can leave a MATLAB DES and enter another MATLAB DES.

Event

Multiple types of events can be scheduled and executed to an entity. These events model activities such as entity creation, destroy, forward (send/receive), delay and search. As of MATLAB/Simulink R2018a, MATLAB DES supports five types of events:

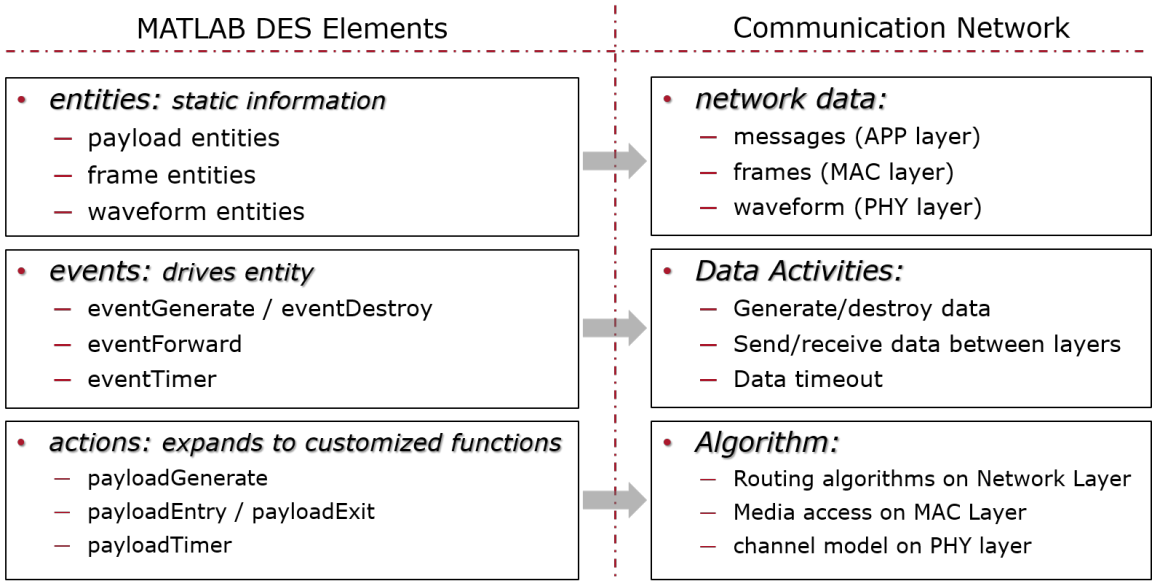


Figure 2.6: Mapping a MATLAB DES Elements to Communication Networks Elements

- Generate: *obj.eventGenerate( )* can generate an entity in the target storage.
- Destroy: *obj.eventDestroy( )* can destroy an entity in the target storage.
- Timer: *obj.eventTimer( )* delays an entity to a period of time.
- Iterate: *obj.eventIterate( )* iterates entities in the target storage with conditions.
- Forward:*obj.eventForward( )* forwards entities to a storage or an output port.

Action

When an event is due for execution, actions are invoked. These actions are conducted by user-defined methods, which may contain the algorithms. This is exactly why the users can create variety of DES models.

A MATLAB DES is suitable to simulate network behaviors, as shown in Figure 2.6. In a network, the data flowing through different network layers can be treated as *SimEvents entities*. For instance, the data unit in the PHY layer is a waveform entity while the data unit in the MAC layer is a frame entity. An entity can be generated, destroyed or forwarded

by different *events*. Users can further define more dynamic behaviors, such as media access on MAC layer, channel model on PHY layer, via *actions*.

2.2.2 MATLAB DES Modeling

MATLAB DES block are located in the SimEvents block library, as shown in Figure 2.7. Users can open SimEvents block library by typing *simevents* in the MATLAB command window. The block in the red circle is the *MATLAB Discrete-Event System* system block. By MATLAB R2018a, MATLAB DES only supports simulation in a combination of the Simulink/MATLAB environment. Executing the DES component in pure MATLAB environment may be addresses in future release.

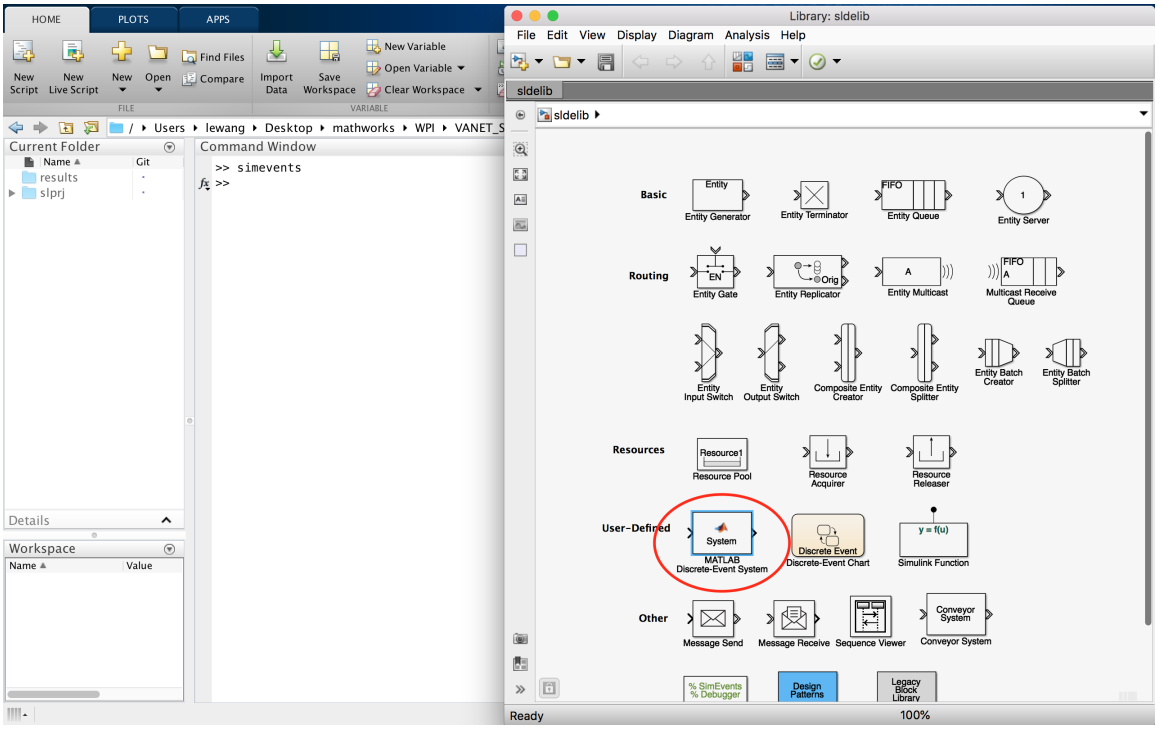


Figure 2.7: SimEvents Library in MATLAB/Simulink R2017b. SimEvents toolbox is a library containing several blocks. Users can open SimEvents Library by typing 'simevents' in MATLAB command window.

Create a new Simulink model and name it as *aSimpleModel*, as shown in Figure 2.8. Drag a *MATLAB Discrete-Event System* block from the SimEvents library to the blank *aSimpleDES* model. Double click the MATLAB DES block, create a new skeleton MAT-

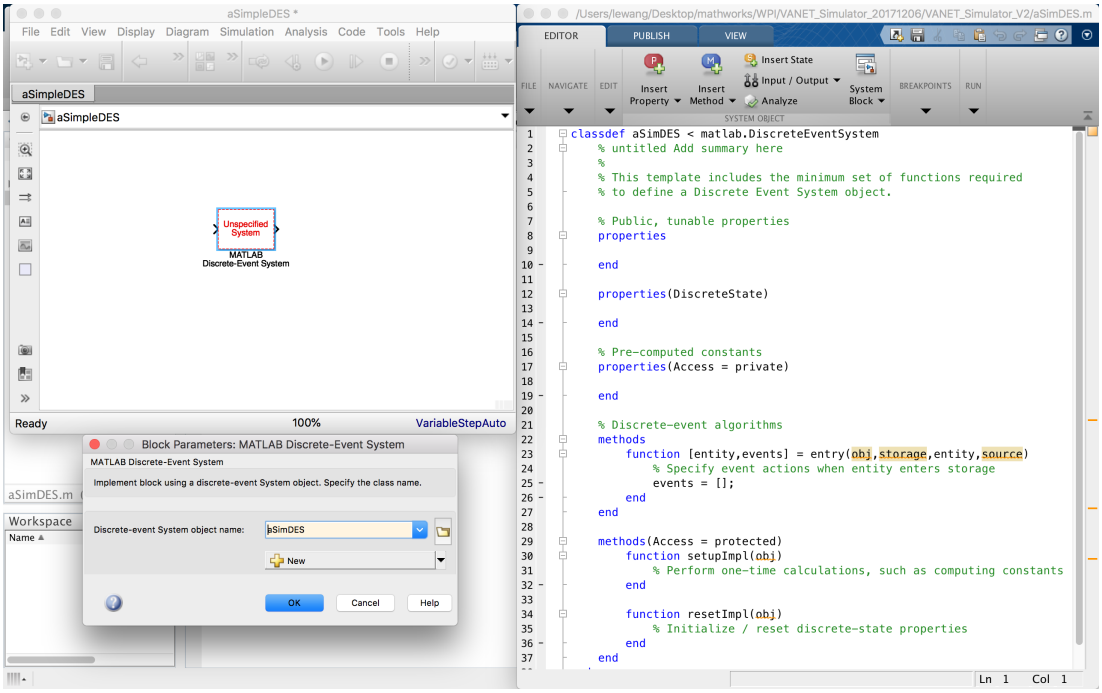


Figure 2.8: The *MATLAB.DiscreteEventSystem* Class in MATLAB and DES block in Simulink. An empty DES system object is created. This DES object is stored in a Simulink system block to work with other Simulink blocks.

LAB DES system object named *aSimDES* that inherits *MATLAB.DiscreteEventSystem*. Select *aSimDES* in the popup list of *Discrete-event System object name*. Then, drag *Entity Generator* block and *Entity Terminator* from the SimEvents library to the model and link them together, as shown in Figure 2.9.

The data flow of this simple DES model is that the *Entity Generator* block generates SimEvents entities periodically, *e.g.,* 10 entities/second. These SimEvents entities pass through *aSimDES* MATLAB DES block. After being processed by the *aSimDES* system object, SimEvents entities leave the *aSimDES* block and enter the *Entity Terminator* block, which destroys all received SimEvents entities.

Notice the major block in the model is *aSimDES* block as the other two blocks are doing simple jobs such as creating or destroying entities. The details of the *aSimDES* block are shown in Figure 2.10. In the figure, *aSimDES* system object contains two storages, *i.e.,* storage 1 and storage 2. When entering the *aSimDES* block, the entities are all stored

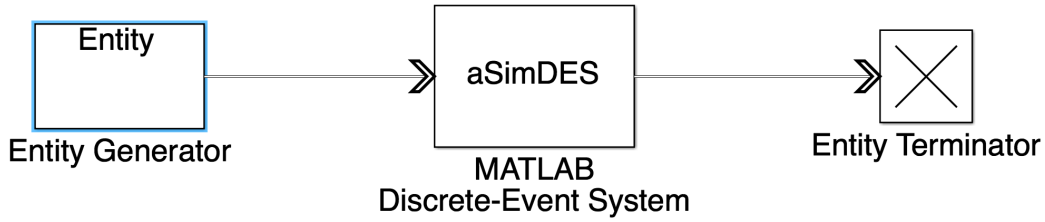


Figure 2.9: A simple Discrete Event System (DES) model in Simulink. The DES model includes an entity generator, a MATLAB DES system block and an entity terminator.

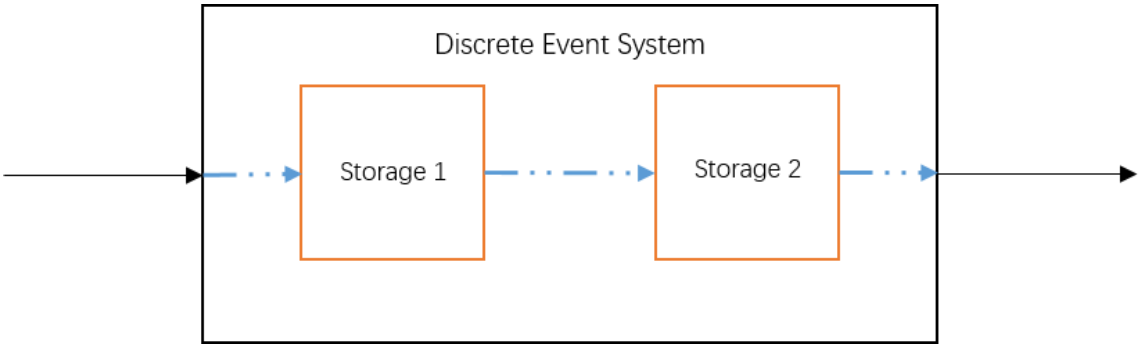


Figure 2.10: The storage details inside the MATLAB DES system block. This DES contains two storages, one storage is connected with input port and another storage is connected with output port.

in *storage 1* as a First-In-First-Out (FIFO) queue. Then, storage 1 forwards the entities to storage 2, which forwards the entities out of the *aSimDES* block via the output port. Storage plays an important role of holding entities. A DES can contain multiple entity storages, with each storage containing multiple SimEvents entities.

Entity Types

In a DES, *entity type* defines a class of entities that share a common set of data specifications (dimensions, data type, complexity) and a common set of methods. One DES can specify multiple entity types, with each type having a unique name. For example, MAC layer receives a *payload* from APP layer, converts it into a *frame* and finally passes it to PHY layer in the format of *waveform*. Accordingly, in the MAC layer DES, three types of entities are defined: payload entity, frame entity and waveform entity. An entity type can

be defined as a static method on class *MATLAB.DiscreteEventSystem* using:

```
function entity1 = obj.entityType(name, dataType);
```

where, the data type of *entity1* is defined by the *name* parameter, where *name* is consistent with *dataType*. *dataType* can be a MATLAB built-in data type, *e.g.,* a numerical data or user-built structured data, *e.g.,* bus. Entity types of a DES are specified by the *getEntityTypesImpl* method. The following codes specify that the MAC layer has three types of entities: payload, frame, and waveform. The capitalized *Payload*, *Frame* and *Waveform* are predefined bus type data. An example of *Frame* bus type data is shown in Figure 2.11.

```
function entityTypes=getEntityTypesImpl(obj)
    entityTypes=[obj.entityType('payload','Payload')...
    obj.entityType('frame','Frame')...
    obj.entityType('waveform','Waveform')];
end
```

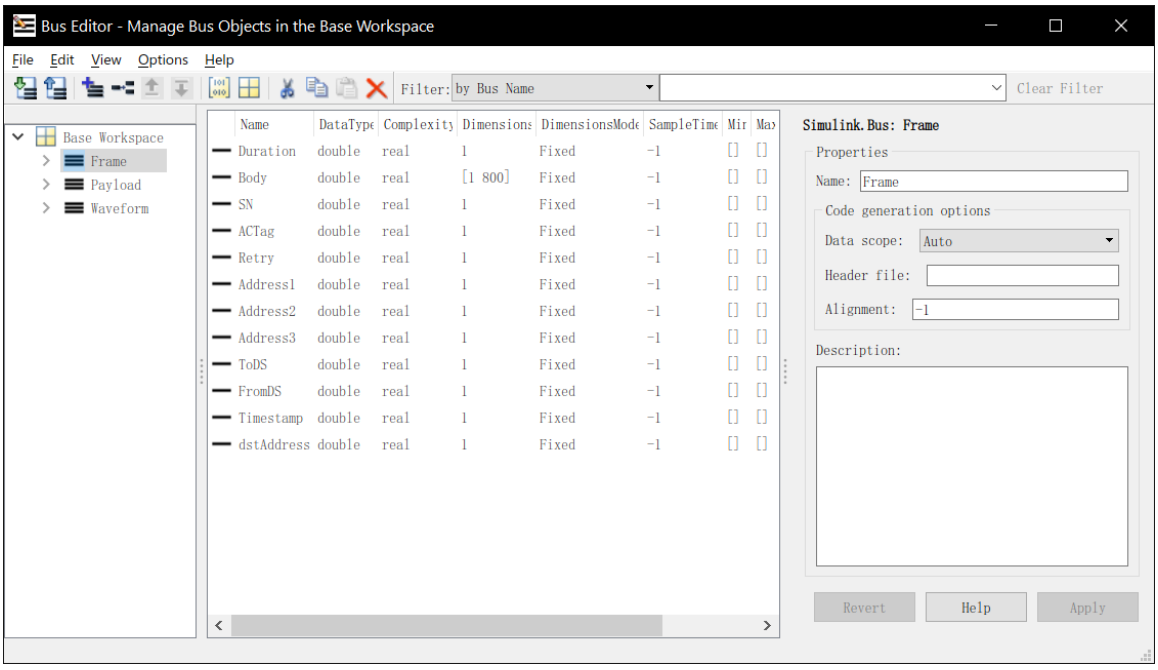


Figure 2.11: An example of *Frame* bus type data. A bus type data contains several elements, each element has some attributes with it.

An entity storage, entity input (I) and entity output (O) must specify the entity type it works with. An entity can be forwarded (i) from an input port to a storage, (ii) from a storage to another storage, or (iii) from a storage to an output port. When an entity *forward* event happens, the source and destination must have the same data type. We start to introduce the definition of an *entity I/O port* and then talk about details on *entity storage* after that.

Entity Ports

As a specific system object, MATLAB DES supports a variable number of input and output ports in the same way as with a general MATLAB system object (via *getNumInputsImpl* and *getNumOutputsImpl*). Users can specify which entity type applies to which entity port. These properties of ports are specified by the *getEntityPortsImpl* method.

For example, the following codes show that a MAC layer DES has two input ports and two output ports. The length of return cell vectors, *i.e.,* $|inputTypes|$ and $|outputTypes|$ must be same as number of input/out ports, *i.e.,* 2 in this example. The input port 1 is a *payload* bus type and input port 2 is a *waveform* bus type because the MAC layer may receive a payload from the APP layer and the waveform from the PHY layer. Similarly, the output port 1 is a *payload* bus type and output port 2 is a *waveform* bus type as the MAC layer converts the received payload to a waveform and sends to the PHY layer. When receiving a waveform from the PHY layer, the MAC layer extracts the payload and sends to the APP layer.

```
function num=getNumInputsImpl(~)
    num=2;
end
function num=getNumOutputsImpl(~)
    num=2;
end
function [inputTypes,outputTypes]=getEntityPortsImpl(~)
    inputTypes={'payload','waveform'};
    outputTypes={'payload','waveform'};
end
```

Entity Storage

A MATLAB DES may contain multiple entity storage. MATLAB DES allows to access an entity at an arbitrary location of the storage, which is similar to the iterator access in a C++ std::list. As of MATLAB R2018a, MATLAB DES storage supports a First-In-First-Out (FIFO) queue, Last-In-First-Out (LIFO) queue, priority queue, and system priority queue. An entity storage is a random-access container with the following properties:

- Storage type: (required) Criteria to sort entities of a storage, such as FIFO queue, LIFO queue or Priority queue.
- Entity type: (required) The type of entity this storage is handling. Note that the *type* here is the entity type defined by *obj.entityType()*, instead of the MATLAB built data types.
- Capacity: (required) Maximum number of entities that the storage can contain. For infinite capacity, use *inf*.
- Key name: (for priority queue, optional) Any name of the attribute used as key for sorting.

- Sorting direction: (for priority queue, optional) Order of sorting: ascending or descending.

The following code shows how to specify the entity storages:

```
storage1 = obj.queueFIFO(entityType, capacity)
storage2 = obj.queueLIFO(entityType, capacity)
storage3 = obj.queuePriority(entityType, capacity, key, order)
storage4 = obj.queueSysPriority(entityType, capacity, order)
```

Users can use the above code to define entity storage via the *getEntityStorageImpl(obj)* method. For example, the following code creates 10 storages as FIFO queues with infinite capacity. The storage units are created with an invisible *storage ID* sequentially, which can be observed from the numbers 1-10 in the code comments below:

```
function [storageSpec,I,O]=getEntityStorageImpl(obj)
    payloadStorage=obj.queueFIFO('payload',inf); % 1-Payload buffer.
    frameStorage=obj.queueFIFO('frame',inf); % 2-frame buffer
    AC0=obj.queueFIFO('frame',inf); % 3-AC0
    AC1=obj.queueFIFO('frame',inf); % 4-AC1
    AC2=obj.queueFIFO('frame',inf); % 5-AC2
    AC3=obj.queueFIFO('frame',inf); % 6-AC3
    HCF=obj.queueFIFO('frame',inf); % 7-HCF
    Txwaveform=obj.queueFIFO('waveform',inf); % 8-Txwaveform
    Rxwaveform=obj.queueFIFO('waveform',inf); % 9-Rxwaveform
    TxFrame=obj.queueFIFO('payload',inf); % 10-to upper layer
    storageSpec=[payloadStorage,frameStorage,AC0,AC1,AC2,AC3,HCF,
        ↪ Txwaveform,Rxwaveform,TxFrame];
    I=[1 9];
    O=[10 8];
end
```

Storage 1 is connected with input port 1 to receive payload entities from APP layer.

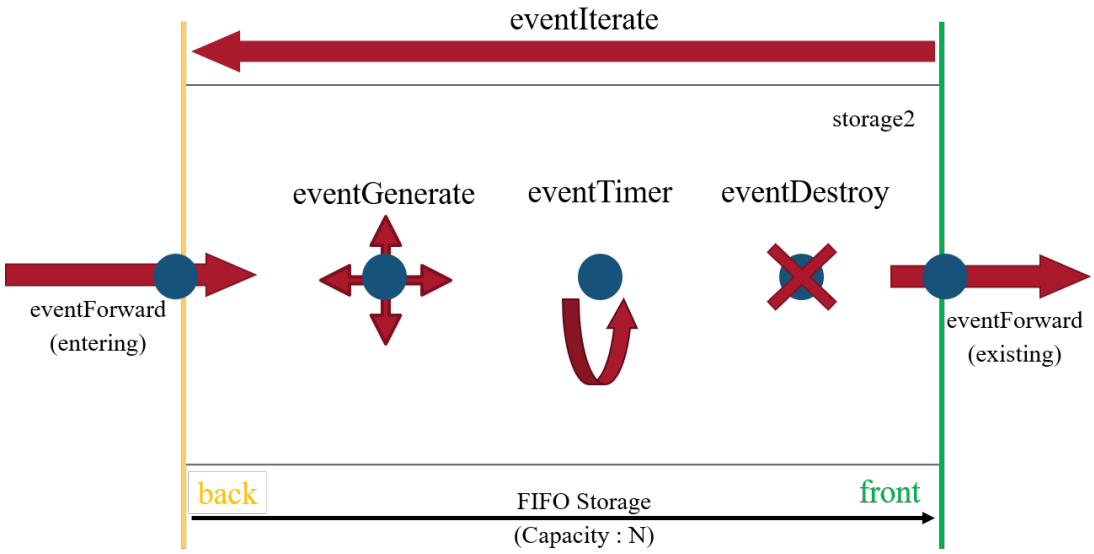


Figure 2.12: The Events and Actions of Discrete Event System (DES) inside one storage. The events include generate, forward, iterate, timer and destroy.

Storage 10 is connected with output port 1 to send payload entities to the APP layer. Both Storage 1 and 2 are of the payload type. Storage 2-7 are of the frame type for the purpose of processing or buffering frame type entities in the MAC layer. Storage 8 connects to the output port 2, which sends the waveform entities to the PHY layer. Storage 9 connecting to the input port 2 is responsible for receiving waveform type of entities from the PHY layer.

One may notice this method also specifies connections between input(I)/output(O) ports and storages. Both I and O are vectors with their length equals to the number of inputs ports or output ports. The nth element in the vector indicates the storage ID that the nth input or output port is connecting to. For example, $I = [1 \quad 9]$ indicates the 1st input port is connecting to storage 1 and the 2nd input port is connecting to storage 9.

Events and Actions

As shown in Figure 2.12, a MATLAB DES has five types of events: *generate*, *iterate*, *timer*, *forward*, and *destroy*. The descriptions of these events are shown in Table 2.3.

When an event is due for execution, *event actions* are invoked as the responses to these events. These event actions are implemented as user-defined methods. Table 2.4 summaries

Table 2.3: MATLAB DES Events and Descriptions.

EVENTS	**TARGETS**	**DESCRIPTIONS**
generate	storage	Create a new entity in the target storage
iterate	storage	Iterate and process each entities in the target storage
timer	entity	Delay the target entity for a certain period of time
forward	entity	Move the target entity from its current storage to another storage or an output port
destroy	entity	Destroy the target entity immediately

each action and their triggering conditions. Details on the complete specifications of events and actions are described next.

Table 2.4: MATLAB DES Actions and Descriptions.

ACTIONS	**TRIGGERING EVENTS**	**DESCRIPTIONS**
generateImpl	generate	called after an entity is created
iterateImpl	iterate	called upon execution of an iterate event.
timerImpl	timer	called after the delay period caused by the timer event
entryImpl	forward	called after an entity enters a storage
exitImpl	forward	called after an entity leave a storage
destroyImpl	destroy	called *before* the target entity is destroyed

Trigger the First Step of a DES

A MATLAB DES is a series of events and actions that are mutually triggered. On the one hand, actions are invoked after the corresponding events happened. On the other hand, more events can be scheduled inside the method of an action. Now the question is, which starts first action or event? As both actions and events are associated with entities, who generates the *first* entity becomes the key point. In order to get the first entity, a DES has

two options:

1. An external entity enters the DES: an *entity generator* is connected to the input port of the DES, as shown in the Figure 2.9. The entity generator creates entities at a constant or variable rate according to the configuration (see Figure 2.13). In the figure, the inter-generation time is a variable *dt*. If *dt* is given a constant value, say $dt = 0.1$, the entities are generated at a constant speed of 10 Hz, *i.e.,* 10 entities/second. *dt* can be further configured with probability model to generate entities in some probability distribution, *e.g.,* Poisson distribution.

2. Generate an entity inside the DES: a DES can generate the first entity inside using *setupEventsImpl(obj)*, which setups some *one-time* events for creating initial entities. These entities can be used to further trigger event actions in other methods. The code below shows how to create the first entity using *eventGenerate()*. Events are often scheduled as the returning outputs arguments of action methods.

```
function [events]=setupEventsImpl(obj) % obj is the currenty DES;
    events=obj.eventGenerate(storageID, tag, delay, priority);
end % eventGenerate() creates the first entity
```

Generate

The 'generate' *event* creates a new entity inside the target storage, and consists of the following parameters:

```
events=obj.eventGenerate(storageID, tag, delay, priority);
```

- *storageID*: ID of the target storage where the new entity will be created.
- *tag*: when creating the same type of entities in the same storage, the activity of event generation can be distinguished by different tags.
- *delay*: the new entity can be generated after a delay period.

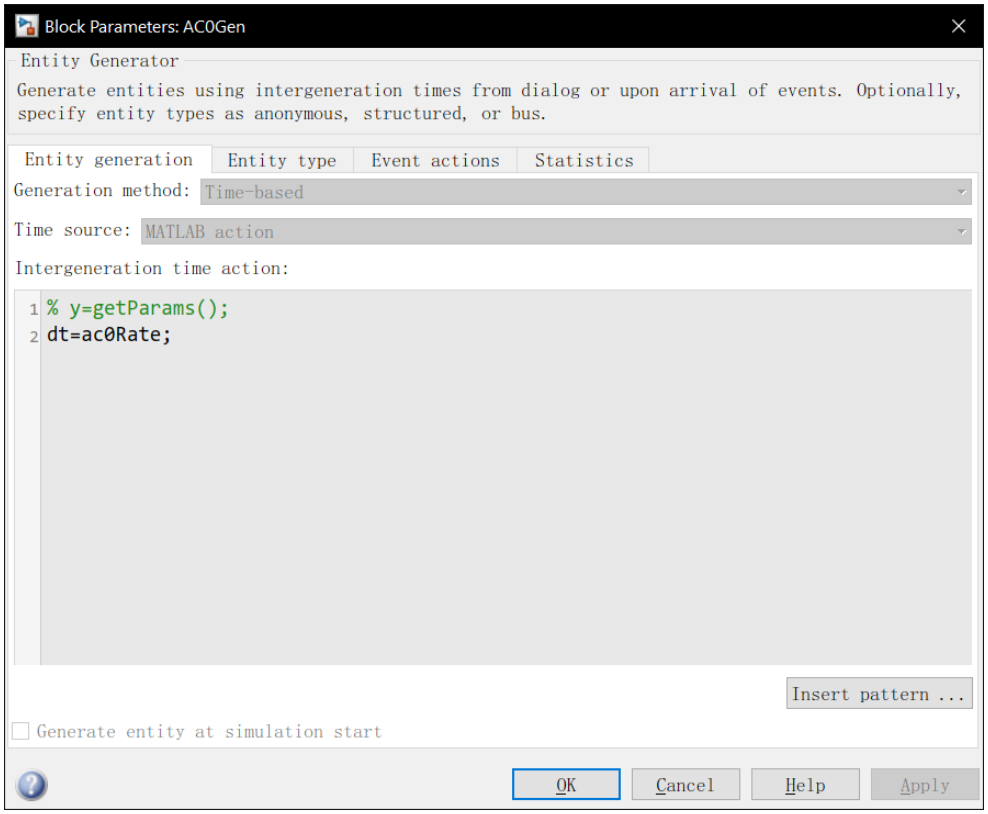


Figure 2.13: The configuration on event generation interval inside an entity generator block. *dt* indicates the entity intergeneration period. *dt* can be a constant or some variable calculated by MATLAB functions.

- *priority*: positive integer value indicating the system priority of the new entity. A smaller value means a higher priority.

The 'generate' *action* is executed upon an entity is created inside a storage, and consists of the following parameters:

```
function [entity,events] = xxxGenerateImpl(obj,storage,entity,tag)
    ...
end
```

- *input parameters:*

  1. *obj*, current discrete-event system.

2. *storage*, the storage where the new entity is created.
3. *entity*, the newly created entity.
4. *tag*, tag of the currently executing 'generate' event.

- *output parameters:*

1. *entity*, entity with possibly changed values after processing by 'generate' action.
2. *events*, events to be scheduled after 'generate' action.

The 'xxx' ahead of *GenerateImpl* indicates the name of an action method is adaptively changed based on the entity types. As for a DES with multiple types of entities, each type of entity shares a common set of event action methods. For example, a DES has two types of entities called 'payload' and 'frame', and both 'payload' and 'frame' entities require *eventGenerate* actions. A simple rule is applied to distinguish the same actions to different entities.

```
Method name = <entity type name> <Action name> Impl;
```

Based on this rule, the *generate* action for 'payload' is *payloadGenerateImpl()* and for 'frame' is *frameGenerateImpl()*.

Iterate

The 'iterate' event iterates and processes each entities of a storage, and consists of the following parameters:

```
events=obj.eventIterate(storageID, tag, [priority])
```

- *storageID*: ID of the target storage where entities inside this storage will be iterated.
- *tag*: when iterating the same type of entities in the same storage, the activity of iteration can be distinguished by different tags.
- *priority*: optional, priority of the entity iterate event.

Upon execution of an Iterate event, the 'iterate' action is invoked for each entity from front to back of the storage, with the option of early termination.

```
function [entity,events,next] = xxxIterateImpl(obj,storage,entity,
    ↪ tag,cur)
    ...
end
```

- *input parameters:*
  1. *obj*, current discrete-event system.
  2. *storage*, the storage being iterated.
  3. *entity*, currently iterated entity.
  4. *tag*, tag of the currently executing 'iterate' event.
  5. *cur*, a MATLAB struct indicating the current iteration state. 'cur.size' indicates total number of entities in the storage; 'cur.position' indicates the position of the current iterating entity.
- *output parameters:*
  1. *entity*, entity with possibly changed values after processing by 'iterate' action.
  2. *events*, events to be scheduled after 'iterate' action.
  3. *next*, boolean value indicating whether the iteration shall continue (true → continue, false → break).

Timer

The 'timer' event delay an entity for a certain period of time, and consists of the following parameters:

```
events=obj.eventTimer(tag,delay);
```

- *tag*: tag of this timer event.

- *delay*: time delay between current simulation time and the time that this timer event will be executed.

The 'timer' action is invoked upon a previously scheduled timer event has expired, and consists of the following parameters:

```
function [entity, events]=xxxTimerImpl(obj,storage,entity, tag)
    ...
end
```

- *input parameters:*
  1. *obj*, current discrete-event system.
  2. *storage*, the storage where the 'timer event' related entities are stored.
  3. *entity*, then entity regarding the timer event.
  4. *tag*, tag of the currently executing 'timer' event.
- *output parameters:*
  1. *entity*, entity with possibly changed values after processing by 'timer' action.
  2. *events*, events to be scheduled after 'timer' action.

Forward

The 'forward' event moves an entity from its current storage to another storage or output port, and consists of the following parameters:

```
events=obj.eventForward(locationType, locationIndex, delay);
```

- *locationType*: the type of the target location, can be either *storage* if the destination is another storage, or *output* if the destination is an output port.
- *locationIndex*: the index of the target location. For example, *(storage,1)* means the first storage, *(output,2)* means the second output port.

- *delay*: delay period before the entity is forwarded to the destination.

The actions corresponding to *forward* event is a little special because the activity of 'forward' involves *leaving an old storage* and/or *entering a new storage*. Sometimes, entering a storage does not even require *forward* event. We start from *entry* action and describe *exit* action after that.

As long as an entity enters a storage, 'entry' action can be invoked no matter where this entity came from. An entity can enter a storage from another storage or form an input port. Recall the section of *Triggering the First Step of a DES* , an external *entity generation* can generate entities and these entities enter the connected DES via an input port and finally enter the storage which connects to the input port. This process is not triggered by 'forward' event.

```
function [events, entity] = xxxEntryImpl(obj,storage,entity,source)
    ...
end
```

- *input parameters:*
  1. *obj*, current discrete-event system.
  2. *storage*, the storage that the entity enters to.
  3. *entity*, then entity entering the storage.
  4. *source*, location where the entity comes from, *e.g.,* an input port or another storage.
- *output parameters:*
  1. *entity*, entity with possibly changed values after processing by 'entry' action.
  2. *events*, events to be scheduled after 'entry' action.

The 'exit' action is invoked *before* an entity exits from a storage. Attention that when an entity exits a storage to an **output port**, this entity will leave the current MATLAB

DES, which is quite similar to the situation when an entity is *destroyed*. Upon an entity left the DES, all its attached unfinished lasting events are immediately invalid and will never happen. The entities leaving to another storage are not affected by the above rule.

```
function [events] = xxxExitImpl(obj,storage,entity,destination)
    % output does not return an entity becuase the enetity is
        ↪ leaving the current DES.
    ...
end
```

- *input parameters:*
  1. *obj*, current discrete-event system.
  2. *storage*, the storage where the entity exits from.
  3. *entity*, the entity existing the storage.
  4. *destination*, the destination that an entity is going to, *e.g.,* an output port or another storage.
- *output parameters:*
  1. *events*, events to be scheduled after 'exit' action.

Destroy

'destroy' event destroys an existing entity of a storage. Attention that when *eventDestroy* is called, the corresponding entity is destroyed immediately. All the unfinished events attached to this entity will also be eliminated at once. Events with 'delay' as the input parameter, such as *eventTimer*, *eventGenerate* and *eventForward*, are called lasting events. It is possible that when an entity is destroyed, its lasting events are in the middle of delay period and not start yet. These events become invalid immediately when the entity is destroyed and will never happen even if the delay period is ended.

```
events=obj.eventDestroy(); % no input parameters.
```

The 'destroy' action is invoked *before* an entity is destroyed, and consists of the following parameters:

```
function [events]=xxxDestroyImpl(obj,storage,entity)
    % output does not return an entity becuase it is to be destroyed
        ↪ .
    ...
end
```

- *input parameters:*
  1. *obj*, current discrete-event system.
  2. *storage*, the storage containing the *to-be-destroyed* entity.
  3. *entity*, the *to-be-destroyed* entity.
- *output parameters:*
  1. *events*, events to be scheduled after 'destroy' action.

DES Flow Chart

In a DES, events and actions are mutually triggered in chains as *action-¿action body-¿event-¿action*. In this dissertation, we proposed a special DES flow chart to describe the events and actions chains, as shown in Figure 2.14. In the figure, the square block indicates an action, while a round block indicates an event. The code between the action and event block is the action body, which expand the action to different activities or algorithms or trigger another event(s).

Figure 2.14 shows partial code when converting a payload entity to a frame entity. Action *payloadEntry* is triggered when a payload entity enters the DES module. Then, the DES module starts to run the codes inside the action *payloadEntry* body. The code here mainly store information from a WAVE short message (WSM) of type 222 to the local variables. Then, an event *eventGenerate* is called to generate a frame entity with a tag of *rcvWSA*. A tag is associated with a specific action because multiple events of same type,

such as *eventGenerate*, but different purposes may be called at the same time. The events are distinguished by their tags. In our case, event *eventGenerate('rcvWSA')* triggers action *mgmFrameGenerate(tag)*. It is possible that another event, such as *eventGenerate('BSM')*, also triggers action *mgmFrameGenerate(tag)* simultaneously. Thus, inside the action body code, both activities are distinguished by tags, one is 'rcvWSA' and another one is 'BSM'. The 'rcvWSA' action creates a new frame entity and defines its type as 1 and assigns multichannel information (MCinfo) to the frame entity's data field. Once the action code of *rcvWSA* is conducted, a new event *eventForward('output',2,0)* is called, which forwards the newly generated frame entity to the 2nd output port of the DES module.

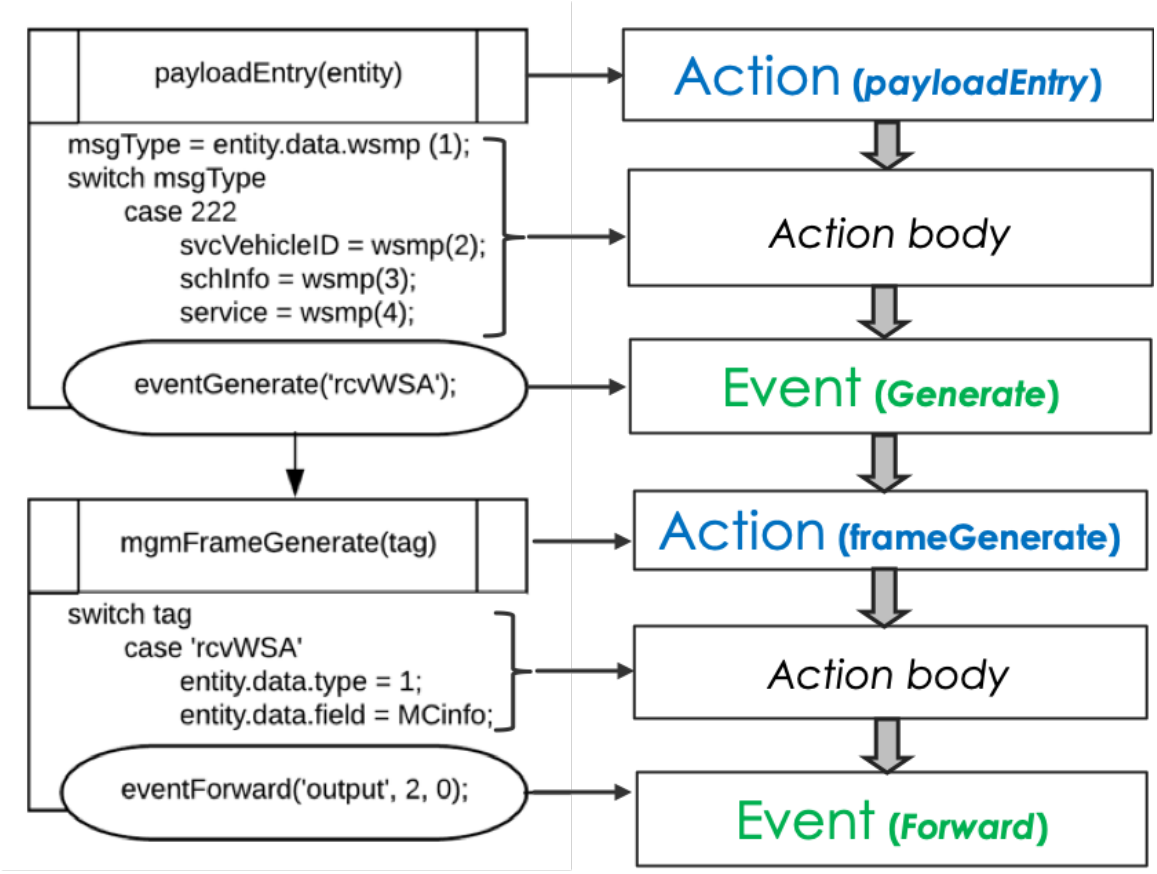


Figure 2.14: The driver process of a discrete-event system (DES). A DES is activated based on the mutually triggered events and actions. When an event is done, an action is triggered. The action body can trigger another event(s).

2.3 Protocols of Vehicular Network

In this section, the WAVE standards are compared with the TCP/IP model. The details of the vehicular network stack are also provided.

2.3.1 Overview of WAVE/DSRC Standards

Intelligent Vehicle Highway Systems (IVHS) have proposed the vehicular network concept in 1991 in order to increase driving safety, decrease traffic congestion, as well as conserve fossil fuel. In reference [23], the US Department of Transportation (DoT) summarized 37 pre-crash scenarios and determined that most of them could potentially be addressed by vehicle communication technology. The document showed the total cost of crashes is $ 274,929,000,000 USD. The US DOT advised the Intelligent Transportation Society of America (ITSA) to created the intelligent transportation system (ITS) in 1996.

In 2004, IEEE approved the IEEE 802.11p standard, which is an amendment to the IEEE 802.11a standard, in order to provide a wireless access in vehicular environments (WAVE) [20,66]. The WAVE defines enhancements to the IEEE 802.11 (the standards used for Wi-Fi) required to support ITS applications based on the ITSA proposals. The IEEE 802.11p defines the Physical (PHY) layer in the licensed ITS band of 5.85-5.925 GHz for the data exchange between vehicles and between vehicles and infrastructures. The 802.11p is the basis for dedicated short-range communications (DSRC), a US DOT project for vehicle-based communication networks. The IEEE 1609 standard suite defines the functions of the higher layer of WAVE, such as the multichannel EDCA operation in the MAC layer and the WAVE short message protocol (WSMP) in the APP layer.

WAVE finally becomes the solution of the vehicular network by using both the IEEE 802.11p and the IEEE 1609 protocol sets. WAVE is designed to support long ranges of operations (up to 1000 meters) among high speeds of vehicles under extreme multipath environments. In WAVE, two types of communication forms are defined, the vehicle to vehicle (V2V) communication [67] and the vehicle to infrastructure (V2I) communication. The V2V communication connects between two or more OnBoard Units (OBUs), while the V2I communication connects between OnBoard Units (OBUs) and RoadSide Units (RSUs)

installed in the roadside infrastructure.

The WAVE vehicles may form a WAVE Basic Service Sets (WBSS) to further share information within a small area network using specific service channels (SCHs), which is similar to the 802.11 basic service set (BSS) [68]. One of the unique characteristics of a WBSS is the establishment process does not require the authentication and association coordinations. Additionally, the WBSS does not rely on a base station for all BSS management. These features are called Outside-of-the-Context of a BSS (OCB) mode, which is very different relative to other 802.11 systems.

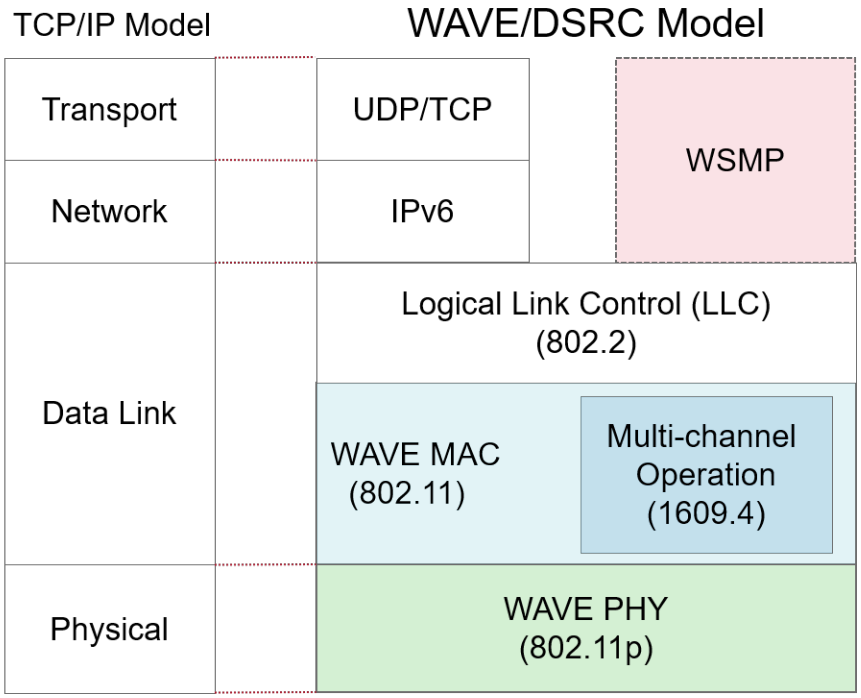


Figure 2.15: WAVE Protocol Stack including the IEEE 802.11p and the IEEE 1609 standards. The colorful fields are the functions that have been implemented in the proposed simulator. This dissertation mainly focuses on the colorful fields.

Figure 2.15 shows the WAVE protocol stack and its correspondence to the TCP/IP network model. The IEEE 802.11p defines the MAC and PHY layers. All the other layers above the MAC layer are defined by the 1609 standards [69]. For example, 1609.4 defines an extra sublayer between the MAC and LLC layers. This sublayer supports the multichannel behavior with EDCA scheme. The link layer and the PHY layer correspond to the same

layers in the TCP/IP model. The WAVE APP layer is different with the TCP/IP APP layer. WAVE supports TCP/IPv6 socket same as the TCP/IP model. However, the WSMP protocol in the APP layer defines a new type of messages, WAVE short messages (WSMs), which is uniquely supported by WAVE. The WSMs contain the basic traffic information and can be broadcast directly without the need of the hand-shake coordination as the TCP does.

2.3.2 Details of Vehicular Network Stack

This section provide a brief overview of the WAVE/DSRC standards including the PHY layer, the MAC layer, and the APP layer.

PHY Layer

The PHY layer of WAVE is derived from the IEEE 802.11a standard and a 10MHz wireless channel is recommended. Table 2.5 compares the parameters between the IEEE 802.11p PHY layer [70] with the IEEE 802.11a PHY layer [71]. Similar to the IEEE 802.11a, DSRC/WAVE uses Orthogonal Frequency Division Multiplexing (OFDM) including 52 subcarriers, *i.e.,*, 48 data subcarriers and 4 pilots subcarriers. For the purpose of decreasing intersymbol interference (ISI) caused by multipath propagation between the high speed vehicles, the IEEE 802.11p bandwidth is downsized to 10 MHz, half bandwidth of the IEEE 802.11a standard. Meanwhile, other parameters including symbol duration, guard time, FFT period and preamble duration are doubled in order to cope with ISI.

MAC Layer

The Media Access Control (MAC) layer is defined in the IEEE 802.11p standard, which is currently integrated with the IEEE 802.11-2012 standard [72]. For the purpose of distinguishing the IEEE 802.11p from other IEEE 802.11 standards, we still use *802.11p* as the indicator of the vehicular network content in the IEEE 802.11-2012 standard for the rest of the dissertation. The default MAC layer in the Wi-Fi devices is using the Distributed Coordination Function (DCF) scheme. The DCF is one type of the MAC layer protocols in

Table 2.5: Parameters of IEEE 802.11a and IEEE 802.11p.

Parameters	IEEE 802.11a	IEEE 802.11p
Modulation	BPSK, QPSK, 16QAM, 64QAM	same as IEEE 802.11a
Code rate	1/2, 2/3, 3/4	
Number of subcarriers	52	
Symbol duration	4 μs	8 μs
Guard time	0.8 μs	1.6 μs
FFT period	3.2 μs	6.4 μs
Preamble duration	16 μs	32 μs
Subcarrier space	0.3125 MHz	0.15625 MHz
Bit rate (Mbps)	6, 9, 12, 18, 24, 36, 48, 54	3, 4.5, 6, 9, 12, 18,24, 27 (half clocked mode)

the IEEE 802.11 standard, which defines Carrier Sensing Multiple Access (CSMA) / Carrier Avoidance (CA) over the contention based channel access.

In order to guarantee the communication quality of a vehicular network, the IEEE 802.11p standard proposed to replace the DCF scheme with the Hybrid Coordination Function (HCF) scheme. The HCF scheme defines the contention-based access scheme and the contention-free access scheme. The latter is using the HCF Controlled Channel Access (HCCA) scheme in order to create a Contention Free Period (CFP), which is similar to the RTS/CTS mechanism in the CSMA/CA protocol. The contention-based access grants data with different level of channel access priorities using the Enhanced Distributed Channel Access (EDCA) scheme. Within the scope of this dissertation, we only focus on the EDCA mechanism in the MAC layer.

In the EDCA mechanism, the data with different importance levels are classified into 8 user priorities (UP) and these 8 UPs are grouped into 4 Access Categories (ACs). Different ACs possess different backoff periods decided by the Arbitration InterFrame Space (AIFS) value and the contention window (CW) value [73]. Table 2.6 shows one set of the AIFS-N/CW parameters and AC0 indicates the lowest priority while AC3 indicates the highest priority. Before sending to the channel, a data is mandatory to wait for a period of time.

Table 2.6: Default EDCA Parameters Set in CCH.

AC	CWmin	CWmax	AIFSN
AC_BK (AC0)	aCWmin	aCWmax	9
AC_BE (AC1)	aCWmin	aCWmax	6
AC_VI (AC2)	(aCWmin+1)/2-1	aCWmin	3
AC_VO (AC3)	(aCWmin+1)/4-1	(aCWmin+1)/2-1	2

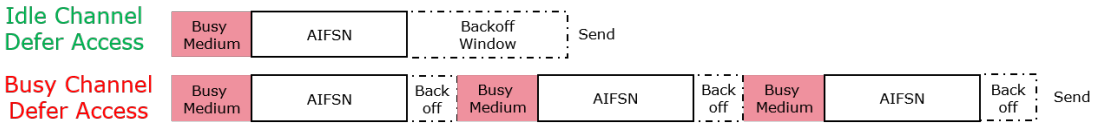


Figure 2.16: The Backoff Delay Comparison between Idle Channel and Busy Channel.

The waiting period is referred to the *channel access deference period*, as shown in Figure 2.16. The figure shows that a channel access deference period consists two sections, AIFS and backoff. When the channel return to the idle state from the busy status, the node will first sense an AIFS period. If the channel keeps idle during the AIFS period, a backoff process begins. If the channel remains idle until the backoff period is finished, a waveform will be sent into the channel immediately. If the channel becomes busy in the middle of a deference period, the node interrupts the current behavior and restarts from sensing an AIFS period. The calculations of the AIFS value and the number of backoff timeslots are explained below.

In the EDCA scheme, each data awaiting to be transmitted is stored into one of the AC queues. The minimum period specifies the idle duration time, AIFS, is no long a constant value as the DCF Inter-Frame Spacing (DIFS) value in the DCF mechanism. AIFS is calculated by Eq. (2.10), in which the slotime and the Short Interframe Space (SIFS) are the predefined fixed values by the IEEE 802.11 standard. Thus the length of the AIFS period varies depending on the AIFSN values:

$$AIFS = AIFSN * slottime + SIFS. \tag{2.10}$$

In Eq. (2.10), the AIFSN is a positive integer varies among different AC queues (see Table 2.6). The slot time is the maximum time period allowing a data to be transmitted between two nodes. In WAVE, the slot time is defined as $13\mu s$. The Short Inter-Frame Spacing (SIFS) is the period between a data and an ACK frame. The IEEE 802.11p defines the SIFS as 32μs. Similarly, the backoff period is calculated by different $[CWmin, CWmax]$ pairs, as described in Table 2.6, and decreased in parallel for each AC queue. CWmin is decided by the aCWmin value, and the CWmax value is calculated from both aCWmax/aCWmin values. EDCA defines the default value of aCWmin as 15 and aCWmax as 1023. At the beginning of the backoff process, the backoff timeslot number is selected uniformly within $[0, CWmin]$. Whenever a retransmission is needed, the CWmin value will be doubled and a new backoff period is calculated within $[0, CWmin]$. The CWmin value is always smaller or equal to the CWmax value.

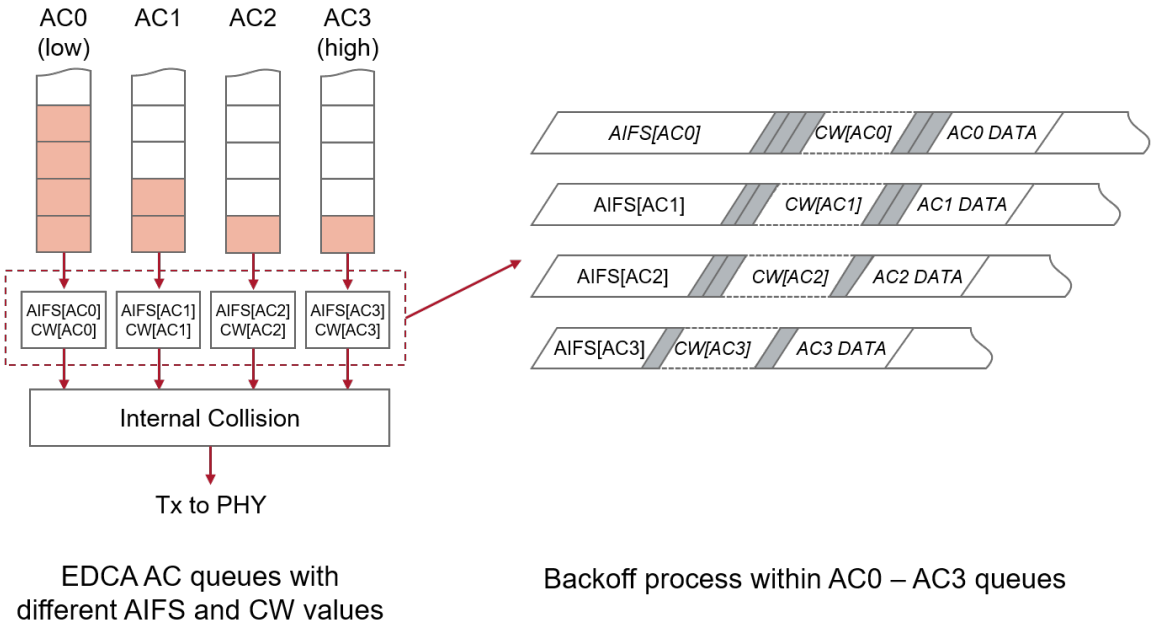


Figure 2.17: The Channel Access Delay for Different AC Queues in EDCA.

Figure 2.17 shows the backoff process with the EDCA scheme. When data enters the MAC layer, it is converted into a frame and forwarded to one of the four AC queues according to its priority. The frames at the head of each AC queue are backing off simultaneously.

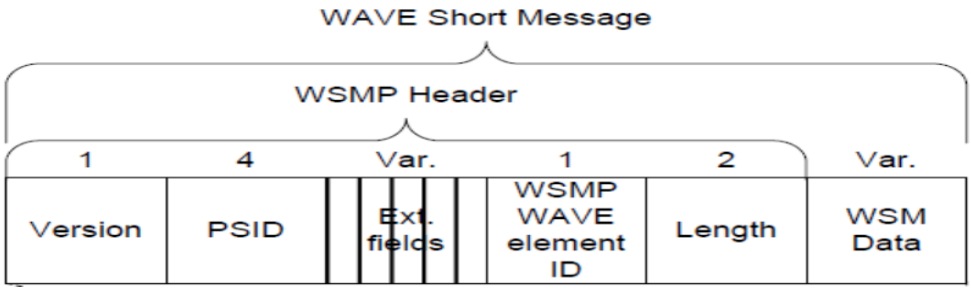


Figure 2.18: Structure of Wave Short Message.

With different [AIFS, CW] values, frames in the lower priority AC queue may experience longer delay than frames in the higher priority AC queue. There is a probability that multiple frames from different AC queues finish their backoff process at the same time, these frames contend for channel access. This contention happens inside the vehicular network node and is called as *internal contention*. When an internal contention appears, only the frame from the higher priority AC queue is allowed to transmit and the other frames with a lower priority increase its retry counter, double the contention window (CW) size, and restart the defer access process. In contrast, the CSMA/CA scheme buffers all frames in the same queue with no different priority levels. A CSMA/CA node only contends for channel access with other CSMA/CA nodes and this situation is called as *external contention*. An EDCA node may experience both internal contention and external contention. The design purpose of the EDCA scheme is to let the data with higher priority gain more channel access probabilities.

APP Layer

The Application (APP) Layer of a vehicular network supports three types of messages: WAVE Short Messages (WSMs), control messages and IP-based messages. The WSMs are designed to consume minimal channel capacity while transmitting safety-related messages. The control messages, are for management operations such as WAVE Service Advertisement (WSA) frame, which is used to establish a WBSS with other vehicles. The IP-based messages are mainly for infotainment purpose and is suitable to be exchanged under high throughput scenario. This dissertation adopts WSMs to transmit both safety-related mes-

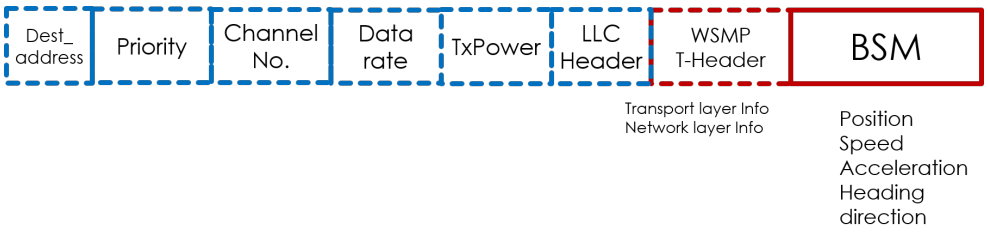


Figure 2.19: BSM Structure in WSM.

sages and non-safety messages. The IP-based messages are not considered in this dissertation.

The WSM protocol (WSMP) is designed for air interface efficiency and low latency in support of vehicular applications. The format of WSMP is shown in Figure 2.18. The channel, transmit power, and data rate are set by higher layers on a per-message basis. WSM is delivered by WSMP to any higher layer entities with interest in the associated *PSID* field.

One of the frequently used safety-related messages is named as the Basic Safety Message (BSM). SAE J2735 [74] defines the format of the BSM to share the position information of the vehicles in the coverage area. BSM is contained in a WSM as shown in Figure 2.19. The BSM contains vehicle driving information including position, speed, acceleration and direction. ITS recommends that the communication delay should be less than 100 ms [75–77] as the BSMs are broadcast beacons at a suggested constant rate of 10 Hz. In order to guarantee the performance of BSMs, the EDCA scheme is suggested to give a higher priority to the BSM as described in [78]. The standard SAE J2945 [79] specifies the minimum communication performance requirements of the SAE J2735 DSRC message sets and associated data frames and data elements. The SAE J2945 standard further defines the BSM transmission rate, transmit power, and adaptive message rate to ensure the interoperability between vehicles.

2.3.3 Concepts of Multichannel Operations

The PHY layer of a vehicular network adopts the Dedicated Shorted Range Communication (DSRC) channel, which is a short to medium range communication service that

supports both public safety and private operations in the V2V and V2I communication. The 75 MHz DSRC spectrum at 5.850 - 5.925 GHz is divided into seven channels of 10 MHz bandwidth, and the 802.11p standard supports 20 MHz bandwidth if needed as shown in the Figure 2.20. Channel 174 and 176 can be combined together to form a 20 MHz channel Channel 175. Similarly, Channel 180 and Channel 182 can form Channel 181 with 20 MHz bandwidth. The DSRC channels include 1 Control Channel (CCH) and 6 Service Channels (SCHs). The CCH is used to transmit control messages as well as safety-related messages, while the SCHs are for the exchange of messages with lower priority.

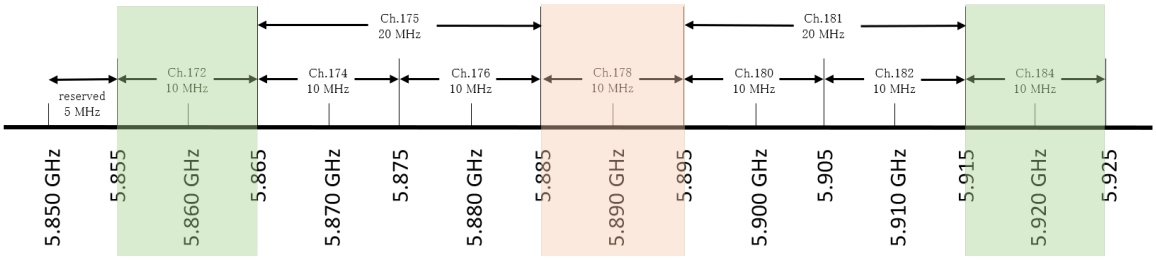


Figure 2.20: FCC Channel Allocation for Vehicular Networks with 1 CCH and 6 SCHs.

When multichannel option is enabled, the channel access time is sliced into synchronization (Sync) intervals. Each Sync interval consists of one 50-ms CCH Interval (CCHI) and one 50-ms SCH Interval (SCHI). The first 4 ms of the CCHI and SCHI is named as Guard Intervals (GI) and reserved for the radio to switch between frequencies. The CCH is activated during the CCHIs and at least one SCH might be activated during the SCHIs. The IEEE 1609.4 standard [80] defines two multichannel options: single-radio multichannel operation and multi-radio multichannel operation. This dissertation only focus on the single-radio multichannel operation. In this scenario, a single-radio device using alternating channel access synchronized to a standard time base. The channel timing is defined such that a Sync interval begins at the start of a second in Coordinated Universal Time (UTC). The UTC is usually provided by GPS. If GPS is not available, the vehicles can synchronize their clock by WAVE Time Advertisement (WTA) frames from other vehicles. If neither of above options are available, the vehicles have to stay in the CCH until their clocks can be synchronized.

In addition to the alternating channel access, the IEEE 1609.4 [80] standard also defines other multichannel access options such as immediate access and extended access. The immediate access scheme allows a user device to switch to the SCH immediately on recognizing a desirable application-service in a WSA. The extender access allows a user device, normally in alternating mode, to access the SCH for an extended period. This dissertation only focus on single radio with alternating channel access scheme.

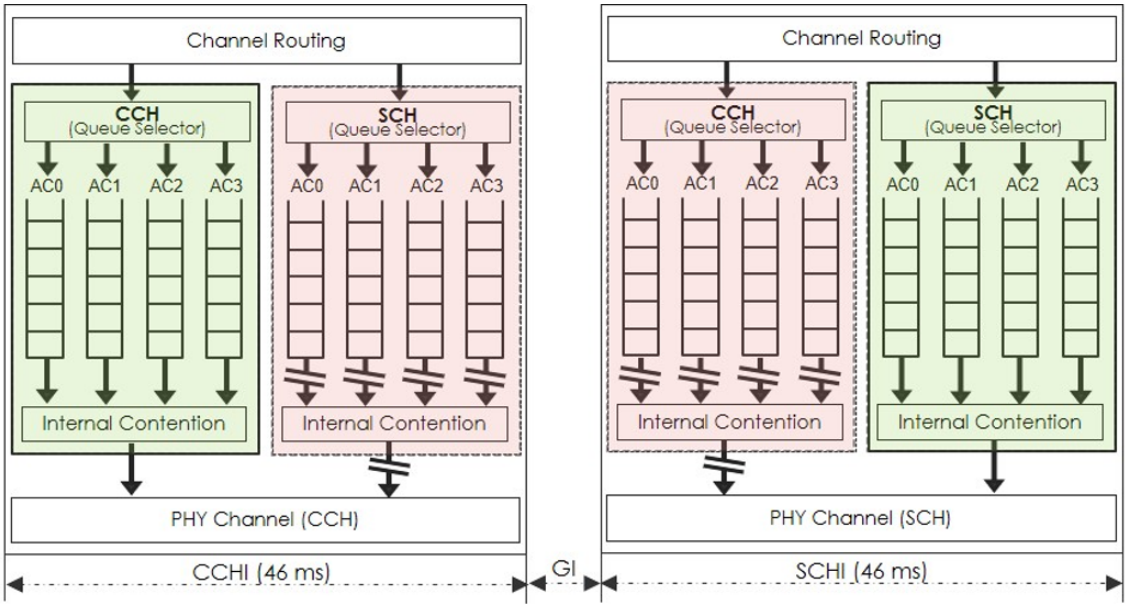


Figure 2.21: Two EDCA modules for multichannel MAC layer.

The multichannel MAC layer is working closely with the multichannel PHY layer, where one CCH and six SCHs are defined. In the single channel scenario, the MAC layer owns one EDCA module with four AC queues. In the multichannel MAC layer, an extra EDCA module is added as shown in the Figure 2.21. In the figure, one EDCA module is for CCH messages and the other EDCA module is for the SCH messages. Both EDCA modules are alternating corresponding to the channel intervals. In the CCHI, the CCH EDCA module is active, the data buffered in the CCH AC queues are contending for channel access. Meanwhile, the activities of the SCH AC queues are suspended. During the SCHI, the activities of the CCH EDCA module are paused and the activities of the SCH EDCA module are resumed. During the GI, both the CCH EDCA and the SCH EDCA modules

are paused.

Table 2.7: The EDCA parameters for CCH and SCH.

Access Category	CCH		SCH	
	[CWmin, CWmax]	AIFSN	[CWmin, CWmax]	AIFSN
AC0	[15, 1023]	9	[15, 1023]	7
AC1	[7, 15]	6	[15, 1023]	3
AC2	[3, 7]	3	[7, 15]	2
AC3	[3, 7]	2	[3, 7]	2

The CCH EDCA module and the SCH EDCA module may have different [AIFSN, CW] values. Table 2.7 shows an example of different EDCA parameter sets for CCH EDCA and SCH EDCA modules. The difference in the EDCA parameters is mainly because the multichannel applications are different. The major applications in the CCH are the safety-related services aiming at providing a short latency and a high packet delivery rate transmission environment for safety-related messages, while the SCH applications are non-safety services aiming at high throughput. The adjustment of the SCH EDCA parameters may potentially decrease the gap between the priority levels in case higher priority services may consume most of channel sources. In the dissertation, we adopts the default suggested EDCA parameters from the IEEE 1609.4 standard as shown in the Table 2.7.

2.4 Chapter Summary

In this chapter, we discussed the classifications of systems. A discrete event system (DES) is defined as a discrete-state event-driven system. A DES can be simulated by timed automata or event scheduling scheme. Then, the basic elements including entity, event and action of a MATLAB DES are introduced. *VANET Toolbox* is a series of more sophisticated MATLAB DES. Building a simple MATLAB DES provides the foundation of building a more complex MATLAB DES. The last section provided a brief overview of the WAVE/DSRC standards. In the next chapter, we will explain the design of *VANET Toolbox* in details based on MATLAB DES and vehicular network theory in this chapter.

Chapter 3

Design Vehicular Network Simulator based on Hybrid DES

This chapter presents the design structure of vehicular network based on a hybrid of time-driven and event-driven systems. Then the implementation of vehicular network simulator as well as peripheral functions including designing connectionless Simulink blocks and databases (local and global) using MATLAB DES is also discussed with details. Further, a tutorial on how to create and run a model using VANET Toolbox is provided in the last section.

3.1 Design of Vehicular Network Simulator

This section describes the design flow of the vehicular network stack including the PHY layer on the bit level, the MAC layer with EDCA module and the APP layer combined with vehicle mobility models. The design pattern in this section can be extended to any discrete-event programming languages.

Generally, a vehicular network simulation is a combination of a time-driven system and an event-driven system. Figure 3.1 illustrates the design structure of two vehicular nodes communicating over a wireless channel. The framework consists of three DES modules: APP Layer DES Module (APP DES), MAC Layer DES Module (MAC DES) and PHY

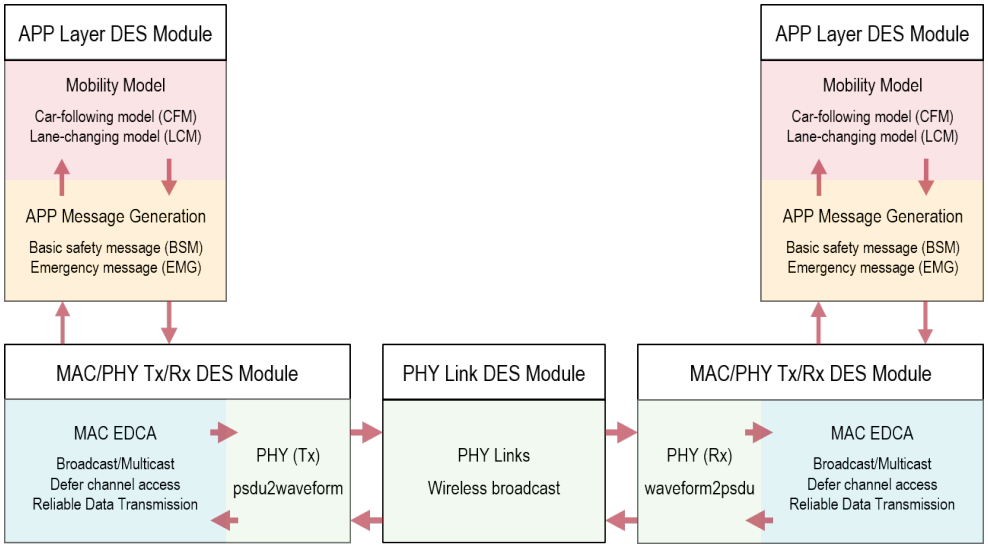


Figure 3.1: Design Structure of VANET Toolbox. The APP layer integrating network model and vehicle mobility model is a hybrid of event-driven and time driven. The MAC layer focuses on EDCA and is purely event-driven. The wireless channel link in PHY layer is a time-driven DES module.

Layer DES Module (PHY Link DES), among which APP DES integrates vehicle mobility models with APP message generation operations, MAC DES includes MAC layer activities and PHY transmitter (Tx) / receiver (Rx) on bit-level processing and PHY Link DES only simulates wireless propagation channels.

The mobility models are integrated in the APP DES, which makes our proposed simulator an *integrated* type simulator. This design facilitates the information exchange between the vehicle mobility activities and the network communication operations. The movements of vehicles are controlled by varieties of mobility models such as car-following model (CFM) and lane-changing model (LCM). The mobility models are implemented by different safety-related or non-safety applications. According to the vehicular traffic scenario, the applications may generate messages and share with other vehicle nodes. This situation is event-driven pattern. Additionally, the applications may create beacon messages such as Basic safety messages (BSMs) at 10 Hz, which is time-driven pattern. Thus the APP DES is a hybrid of event driven and time driven. These generated messages are disseminated via wireless network communication and reciprocally the performance of network communica-

tion could affect the vehicle operations. The section only focuses on the design of network communication simulation environment, the discuss of the mobility models is presented in Section 3.2.3.

The MAC layer of a vehicular network is different compared to other WLAN devices since it grants priorities to various messages such that the messages with higher priority have shorter deference in channel contentions. This mechanism is referred to as Enhanced Distributed Channel Access (EDCA) and it is defined in the IEEE 802.11 standard [81]. Furthermore, the MAC layer is responsible for generating frames, waiting for ACKs, and initiated retransmissions when timeouts occur. All of these MAC layer behaviors are event-driven. In addition, the transmitter (Tx) and receiver (Rx) of PHY layer are integrated with the MAC Layer DES Module. The PHY (Tx) is responsible for converting the binary message information into wireless waveform symbols, while the PHY (Rx) is for the reverse process. Both operations from the PHY Tx and Rx are based on bit-level processing, *i.e.,* users can manipulated every single bit of data when necessary. The integration design of MAC DES has two purposes. First, the PHY operations on bit-level processing is continuous instead of event-driven, thus the PHY Tx/Rx cannot be implemented by DES. In our proposed simulator, a series of functions are created to perform the bit-level processing operations. The second reason is to constrain the total number of DESs in the simulation model in consideration of simulation efficiency. The creation of a DES involves overhead computational costs including assigning input/output ports and allocating queue memories. These overheads may potentially lower the simulation speed. Thus in our design process, one of the most basic requirements is to use as few DES as possible.

The PHY link DES module only simulates the wireless channel links since both PHY Tx and Rx are integrated with the MAC DES module. The PHY link is a relatively easy DES, as it is only responsible for accepting the incoming waveforms from the PHY transmitters and forwarding them to the PHY receivers after an air propagation delay. This process is an event-driven pattern. During the air propagation delay, channel models such as AWGN and two-ray ground reflection model can be applied to the waveforms and this progress is implemented by functions instead of DES.

3.1.1 PHY Layer Design

In vehicular network simulations, a precise representation of the PHY layer is necessary in order to obtain reliable results for comparison with real hardware performance. Popular vehicular network simulation tools including VEINS [82] and iTETRIS [83, 84], which usually have a simplified PHY layer. The network simulators they adopted, NS-3 and/or OMNet++, employ abstracted PHY layer [85], where the smallest indivisible data unit used is the packet, *i.e.,* the packet is either received entirely or not at all. Several details of wireless communication, such as channel estimation, frequency offset estimation and correction, waveform modulation and demodulation, are omitted due to this abstraction. However, individual bits inside a waveform are necessary to perform accurate simulations of the PHY layer and channel models. In this section, we will introduce the proposed PHY layer with bit-level processing techniques. The performance of the PHY layer in terms of packet success rate (PSR) is evaluated in Chapter 4.

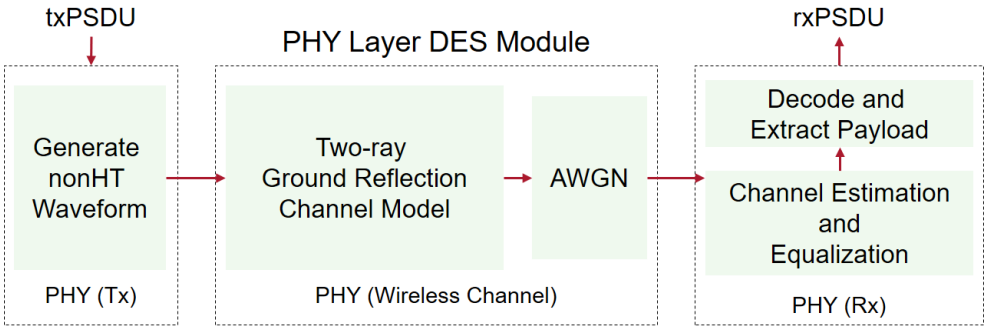


Figure 3.2: VANET PHY Link Modeling. The PHY Tx and Rx are for the data encoding and decoding on bit-level processing. Wireless channel link is a DES module with two-ray ground reflection model and AWGN as default.

Figure 3.2 illustrates a basic wireless PHY link model that converts the received frame from the MAC layer into a wireless waveform and lets the waveform pass through the wireless channel. The interaction between the PHY layer and the MAC layer is handled by the Physical Layer Convergence Protocol (PLCP). A PLCP Service Data Unit (PSDU) is generated by serializing the MAC layer frame into a binary bit stream. The PSDU bits along with the PLCP preamble and header according to the IEEE 802.11 [81] are grouped

into symbols and finally becomes a waveform. The wireless channel consists of a two-ray Ground Reflection Channel model and an Additive White Gaussian Noise (AWGN) model by default. This design is based on the research of line-of-sight (LOS) conditions specified in [86]. Additional channel models including *Rural_LOS* and *Urban_NLOS* can be selected during the simulation. The received waveform is decoded and verified using the bit-level receiver design in [48]. Only the PHY channel link is designed in DES, both PHY Tx and Rx are implemented by functions from WLAN Toolbox and are integrated with the MAC DES module.

Based on [86], the Line of Sight (LOS) channel for vehicular network environment is Two-Ray Ground Reflection model. The Two-Ray model can be found in *Phased Array Toolbox* or programed using the code from [87] attached in Appendix C. The code takes the distance of Tx and Rx antennas, antenna height and transmission power as inputs and outputs the receiving power in dBm. The thermal noise of 10 MHz band is $-80dBm$. According to IEEE 802.11-2012, an implementation loss of $5dB$ is considered, with the noise figure of $10dB$, the total noise is around $-89dBm$. The Signal to Noise Ratio (SNR) can be calculated by $SNR = rcvPower - noise$. Thus, the two ray model can calculate SNR adaptively based on the distances between Tx and Rx antennas, roughly the Euclidean distance of vehicles.

Figure 3.3 illustrates the process of generating a waveform at the transmitter (Tx) at the bit level. The IEEE 802.11p PHY layer is derived from the IEEE 802.11a Non-HT transmission specifications. In MATLAB, *wlanNonHTConfig* creates a Non-HT object in order to configure the transmission parameters. It is configured for a $10MHz$ channel bandwidth with a single transmit antenna according to the IEEE 802.11p standard [81]. A Non-HT Orthogonal Frequency-Division Multiplexing (OFDM) symbol consists of 64 sub-carriers, the $10MHz$ bandwidth decides the symbol period is $6.4\mu s$. A $1.6\mu s$ guard interval (GI) is inserted between each symbols in order to prevent inter-symbol interference (ISI). A PLCP header including information of data rate and PSDU length is prepended to the PSDU. From the perspective of a waveform, the PLCP header is the Legacy Signal (L-SIG) field and the PSDU along with a tail and padding becomes the data field. Ahead of the L-SIG field, a PLCP amble is attached, which includes a Legacy Short Training

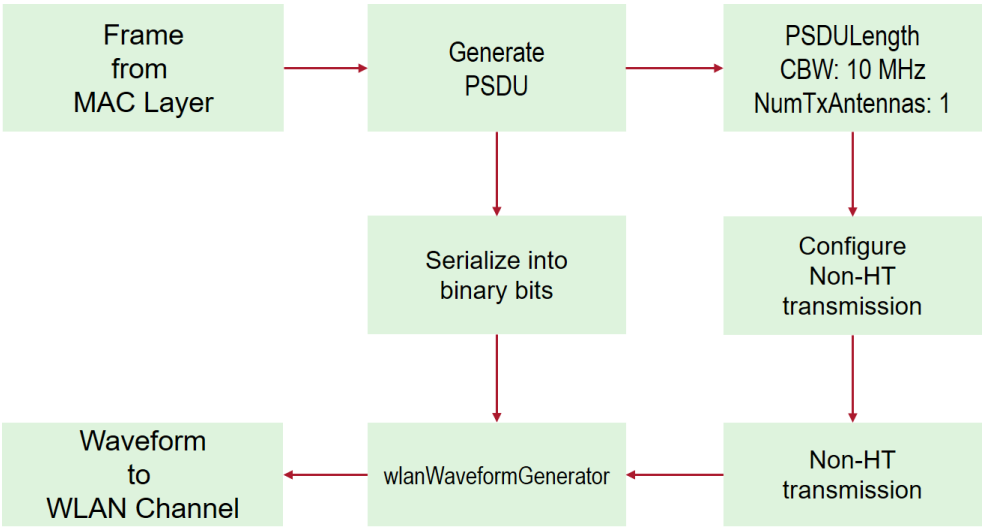


Figure 3.3: Tx: Frames to Waveforms Conversion using MATLAB WLAN Toolbox.

Field (L-STF) and a Legacy Long Training Field (L-LTF). L-STF is used for the packet detection, initial frequency offset estimation, and coarse timing synchronization. The L-LTF is used for the fine time synchronization, channel estimation, and fine frequency offset estimation. Thus a complete waveform consists of a L-STF, a L-LTF, a L-SIG as well as a data field. These fields are generated separately and concatenated to form a complete Non-HT transmit waveform. The Non-HT configuration object specifies the parameters for generating the data fields of a waveform. The *cfgnonHT.PSDULength* property indicates the length of bytes to be sent in the Non-HT data field. A Non-HT waveform is then generated by function *wlanWaveformGenerator* according to the configuration of *wlanNonHTConfig*.

Figure 3.4 shows the process of payload extraction when a waveform arrives at the receiver (Rx). The first field needs to be processed is the L-STF. In the vehicular network, L-STF has a length of $16\mu s$ with 10 repetitions. Due to its good correlation properties, the first seven repetitions are used for the time synchronization purpose by performing self-correlation calculations [88]. The rest of sequence are used for packet detection, coarse frequency offset (CFO) detection and correction [89] and setting the automatic gain control (AGC). The second field needs to be examined is the L-LTF, which is composed of a cyclic prefix (CP) equaling to the period of two GIs, *i.e.,* $3.2\mu s$ followed by two identical long

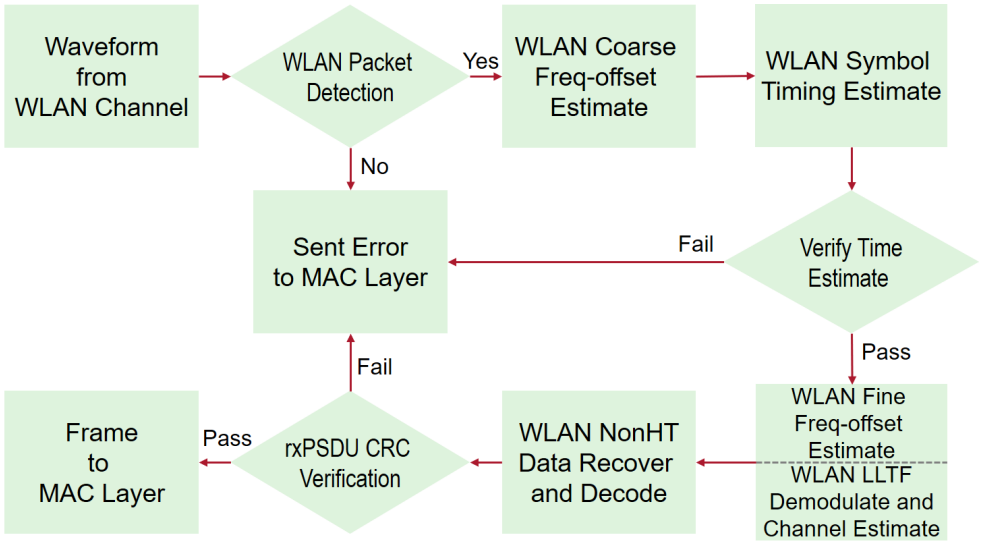


Figure 3.4: Rx: Waveform to Frame Conversion using WLAN System Toolbox.

training symbols, *i.e,* $2 \times 6.4\mu s$. Channel estimation, fine frequency offset estimation, and fine symbol offset estimation all rely on the L-LTF. With all estimation and correction stages executed, the L-LTF demodulator and channel estimator operations are performed based on the demodulated L-LTF. Note that the demodulated L-LTF is also used for noise power estimation. Finally, the Non-HT data field is extracted and recovered into the PSDU. The integrity of received PSDU, *rxPSDU*, is verified by the Cyclic Redundancy Check (CRC). Consequently, the rxPSDU is sent to the MAC layer if it passes the CRC. The above operations of both PHY Tx and Rx are based on the bit-level processing features. This is exactly the same process when an actual waveform is transmitted among radio hardwares [90]. Thus the PHY layer in our proposed simulator is more realistic and accurate.

3.1.2 MAC Layer Design

Vehicular network supports vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication in general. The implementation of MAC layer should be able to cope with all communication mode. For example, after receiving a frame from other nodes, the MAC layer should find out if it comes from another peer vehicle or infrastructure. Besides, the MAC layer should also check the type of the received frame, *i.e.,* broadcast, multicast or

unicast, and prepare for ACK to unicast type frames in reliable data transmission (RDT)

One fundamental difference of IEEE 802.11p when compared to other types of IEEE 802.11 networks is the usage of EDCA for the purpose of Quality-of-Service (QoS). Different frames are granted with different priorities. Eight priorities are defined and can be placed in four possible Access Categories (ACs): AC0, AC1, AC2 AC3. Each frame is assigned one of the AC descriptions by the application that created the message depending on the importance and urgency of the content. Specifically, AC0 denotes regular access, AC1 is for non-prior background traffic, while AC2 and AC3 are for prioritized messages, *e.g.,* critical safety messages.

IEEE 802.11 channels are all contention-based, where all nodes need to compete with each other for channel access. During the contention process, the data is required to wait for a random period of time prior to transmitting, which is referred to as *defer access*. The defer access process includes an Arbitration InterFrame Spacing (AIFS), which is a replacement for Distributed coordination function(DCF) InterFrame Space (DIFS), and a backoff period, which is calculated based on a contention window (CW) value. After sensing a busy medium, a node will wait for an AIFS period before sensing the channel again. If the channel is idle, the node will start to backoff, otherwise the node has to wait for another AIFS period. During the backoff period, the node keeps monitoring the channel status. In the event that a busy channel is detected, the node will immediately pause the backoff and restart the AIFS channel sensing step. In short, both AIFS and backoff define the waiting period for a node before accessing the channel.

In EDCA, the ACs decide different (AIFS, backoff) pairs. Therefore, the frames with different priorities own different defer access periods. In general, the higher priority the shorter the defer period and vice versa. The design purpose of EDCA is to enable the frames the with higher priority to gain channel access more frequently.

In order to depict more clearly the implementation of the MAC layer, we define the data flow from the PHY layer to the APP layer as the inbound flow, as shown in Figure 3.6, and the flow from the APP to the PHY layer as the outbound flow, as shown in Figure 3.5.

For the outbound flow, a payload from the APP layer is converted into a frame by adding the necessary MAC layer headings, then forwarded to the AC0-3 queues. Frames

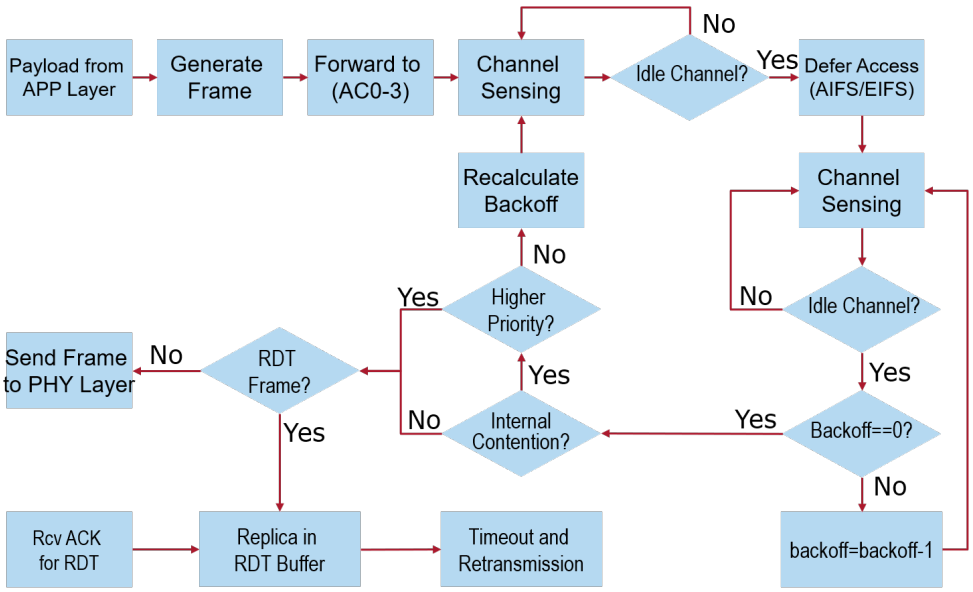


Figure 3.5: MAC Layer Outbound: Data Flow From APP to PHY Layer. A payload is received from the APP layer, converted into a frame, experience channel access backoff and finally converted into a waveform.

from the four AC queues will perform deferred access simultaneously. It might be possible that more than one of the AC queues has frames the are ready to send after the deference period, thus a contention is created. Since this situation happens inside the same node, this type of contention is called an *internal contention*, which is unique for nodes using EDCA. Whenever an internal contention occurs, the frame with the highest priority will be the first one to be sent out, while the other frames have to redo the defer access.

If a frame is of unicast type and requires an ACK from the receiver, *i.e.,* Reliable Data Transmission (RDT), a replica is created inside the buffer and it waits for the ACK. If the ACK is not received within a predefined time period, the replica will be sent again until an ACK is received or the maximum retransmission limit is reached. If the ACK is still not received by then, this frame will be dropped.

For inbound flow as shown in Figure 3.6, a waveform is received from the wireless channel of PHY layer. The receiver (Rx) of PHY layer is integrated within MAC layer DES. Upon receiving a waveform, Rx will perform a series of checks in order.

1. CRC: check if the received waveform is corrupted. If it is corrupted, the waveform is

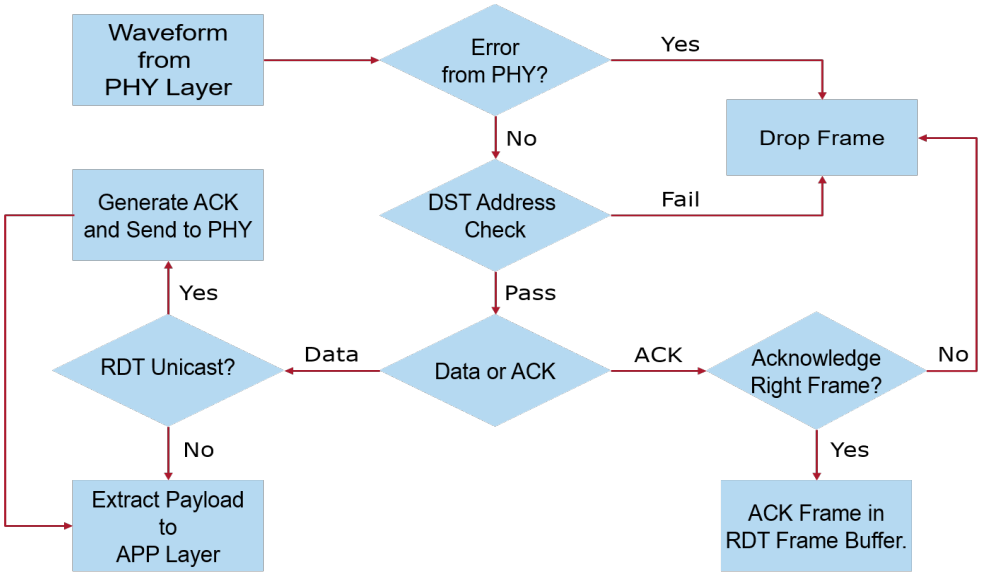


Figure 3.6: MAC Layer Inbound: Data Flow From PHY to APP Layer. Drop the corrupt frame entity, or extract the payload from intact frame and send to the APP layer. Reply an ACK frame if necessary.

detroyed by *obj.eventDestroy*. If the waveform is intact, start the next check.

2. Source and Destination Check: In V2X communication, a network node may be a vehicle or an infrastructure. When a network node receives a waveform, it should check if the waveform is sent to a vehicle node or infrastructure node. Then it will check if the waveform is from an infrastructure or vehicle. This behavior is conducted by checking the *fromDS/toDS* fields in a waveform header.

   Wireless communication is a physical broadcast, *i.e.,* a vehicle may receive a lot of waveforms that are designated to infrastructures. Similarly, an infrastructure may receive many waveforms designated to a vehicle node. Furthermore, in a multi-hop V2X communication environment, a waveform may pass though several nodes (vehicles or infrastructures) before finally reaching to the target node. *FromDS/toDS* only takes 1 bit (2 bits for both) in a waveform hearder field. Checking *FromDS/toDS* is the first barrier to filter large amount of irrelevant waveforms. All these irrelevant waveforms are destroyed immediately without further looking into other information with minor computing resources cost.

3. Address Check: Rx should detect if the received waveform if to the right node by checking the destination address (dstAddress). The dstAddress indicates if this waveform is a broadcast, multicast or unicast waveform. If it is a unicast waveform, the source address (srcAddress) is used to prepare for an ACK. IEEE 802.11 defines 4 address fields in the header, but three of them are actually used in vehicular networks.

   The combination of *ToDS/FromDS* and address fields can guarantee the waveform can be finally delivered to the correct target with minimum resource cost. The *ToDS/FromDS* and address fields are showing in Table 3.1, in which DA refers to destination address and SA refers to source address. BSSID refers to the basic service set ID and can be used to identify different WBSSes. As vehicular network can work in 'outside the context of a BSS (OCB)' mode, BSSID is not mandatory.

Table 3.1: The ToDS/FromDS and Address Fields.

	ToDS	FromDS	Address1	Address2	Address3
V2V	0	0	DA	SA	BSSID
To Infra.	1	0	BSSID	SA	DA
From Infra.	0	1	DA	BSSID	SA

4. Data Type Check: The type of a waveform could be a data or an ACK. If it is an ACK, the MAC layer needs to make sure if it is an valid ACK as multiple ACKs to the same data may be received due to the congestion of channel. If it is a data type waveform, the MAC layer extracts the payload and sends to the APP layer. If it is a RDT waveform, *i.e.,* an ACK is required, the MAC layer will generate the corresponding ACK and send to PHY layer.

Partial code of the above Rx mentioned behaviors is listed below:

```
[status,outframe,typeField,subtype]=waveform2psdu(waveform.Body);
if status==1 %Received correctly
if (ToDS==0 && FromDS==0 && ismember(obj.txAddr,rcvAddress))... %
    ↪ V2V unicast packet
|| (ToDS==0 && FromDS==0 && Address1==0 && Address2~=obj.txAddr)...
    ↪ % V2V broadcast packet
|| (ToDS==0 && FromDS==1 && Address1==0 && Address2==obj.
    ↪ infraAddress && Address3~=obj.txAddr)... %V2I broadcast
    ↪ packet
|| (ToDS==0 && FromDS==1 && Address1==obj.txAddr && Address2==obj.
    ↪ infraAddress && Address3~=obj.txAddr) %V2I unicast packet
switch typeField
case 1 % Rcv ACK: type -> 1 subtype -> 13
...
case 2 % Rcv data: type -> 2
...
end
else % Invalid ToDS/FromDS/Address
events=obj.eventDestroy();
end
elseif status==0  % Corrupted
events=obj.eventDestroy();
end
```

3.1.3 APP Layer Design

One significant challenge of implementing this proposed simulation model is that the behavior of the APP layer depends on the application itself and there is no comprehensive standard defining all the application requirements since it is almost impossible to anticipate

all possible future needs. Therefore, within the scope of this dissertation, the safety-related messages consists of AC2 Basic Safety Messages (BSMs) at a constant 10 Hz and AC3 critical safety messages either from mobility models or at a Poisson distribution. In this section, we introduce the basic message dissemination functions.

A Dedicated Short Range Communication (DSRC) device is required to transmit at least 300 meters [19, 91], and it is assumed that the surrounding vehicle positions are changing frequently in the highly dynamic environment. Consequently, we can assume the safety messages are physically broadcast using a single hop. Therefore, packet collisions and packet loss are major challenges for communication system performance. One solution is to decrease the channel load by grouping similar messages together, as shown in Figure 3.7.

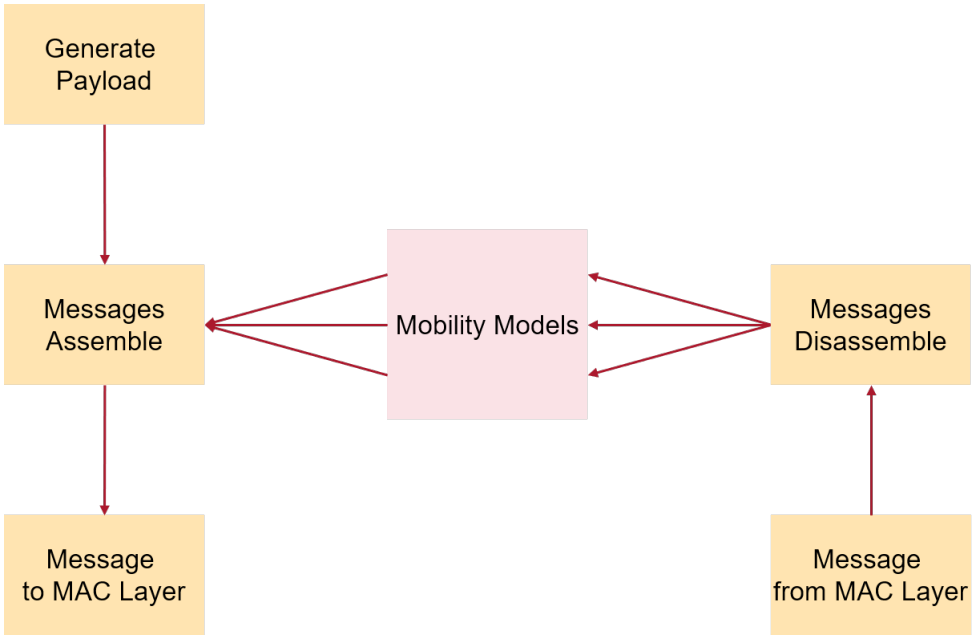


Figure 3.7: APP Layer Design using MATLAB DES. The messages from mobility model applications are converted into payloads and sent to the MAC layer. When receiving payloads from the MAC layer, the messages are extracted and dispatched to different mobility models.

A mobility model may involve several safety applications, such as lane changing, braking [92], and collision warning [93]. These applications share different types of messages with other peers differentiated by application IDs (AppIDs). For example, a lane changing application creates messages that include driving direction information, while braking ap-

plications generate messages containing brake status. While both types of messages may contain the same information such as currently location and speed. If these two messages are sent separately, the overlapping information will cause a waste of transmission source. The APP layer DES maintains a message list created by map containers. Whenever an application is activated, it has to register its AppID as well as its message requirements to the message list. The AppID serves as keys while the requirements are values.

Once an empty payload is generated, the APP layer assimilates the data requirements from these applications and compiles them into one single message using a dictionary of standardized message construction guidelines. In the last example, an assembled message consisting of position, speed, driving direction, and brake status is created instead of two separate messages. Society of Automotive Engineering (SAE) standards J2735 [74] and SAE J2945 [79] define a dictionary with over 150 data elements. Each data element can be indexed using the AppIDs.

When a payload is received, the APP layer is responsible for separating the data elements according to its appID and dispatches them to the corresponding applications so as to finally affect the mobility models.

Mobility Models and Scenarios

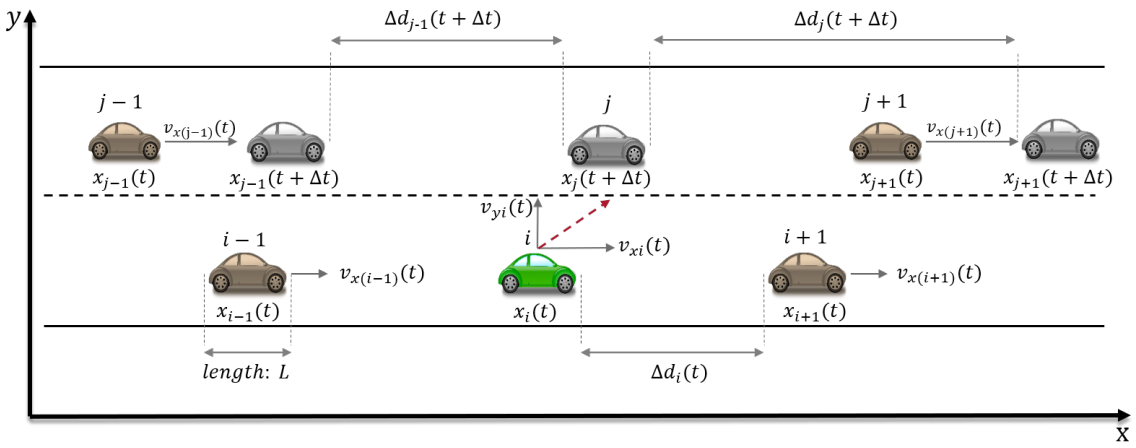


Figure 3.8: Notations for Highway Mobility Model based on V2V Communications.

Research in [94] used the OpenStreetMap for extracting road network into a XML file. Then the traffic mobility flow is read by SUMO to simulate vehicular motions. During the process, several applications and simulators are involved via separate interface programs. For instance, *eWorld* converts the format of the OpenStreet log file into the SUMO readable format, TraNS passes the SUMO road network data to NS-2. Due to these interface programs, it is difficult to conduct simulations in real-time. An integrated vehicular network simulator usually combines vehicular mobility model and vehicular network model together. A proper vehicular mobility model is necessary to reflect the real vehicular traffic behaviors as vehicular mobility impacts the vehicular network performance significantly. A vehicular network application is designed to make use of shared traffic information among vehicles in order to change traffic patterns, either for the purpose of road safety or for the road efficiency improvement.

Therefore, a vehicular network simulator should show the interaction between the network protocol and vehicular mobility. In *VANET Toolbox*, the vehicular mobility models are integrated with the APP layer. Variety of vehicular mobility models have been proposed for different purpose including random models, flow models, traffic models, behavioral models and trace-based models. Traffic safety applications usually requires traffic flow modeling, in which the interactions between vehicles are modeled with details as flows, shown in Figure 3.8.

Vehicular network applications can be classified into V2V applications and V2I applications depending on which V2x mode is used. According to [32], V2V applications are generally safety applications and V2I usually dedicates to traffic efficiency improvement. In this paper, we only focus on V2V communication and two V2V-based mobility models, car following model and lane changing model, are discussed in the next chapter.

3.2 Implementation of Vehicular Network Simulator

In this section, we present the self-designed vehicular network simulator: *VANET Toolbox* using MATLAB Discrete-Event System (DES). *VANET Toolbox* is a Simulink library containing several blocks, as shown in Figure 3.9. The upper section (purple) consists of

blocks for the main layers of vehicular network stack, *i.e.,* Application (APP) layer, Media Access Control (MAC) layer and Physical (PHY) layer.

VANET Toolbox is an *integrated* type simulator, *i.e.,* the simulator includes both vehicular mobility models and vehicular network simulator. In *VANET Toolbox*, the mobility models are integrated with APP layer. As of *VANET Toolbox V2.0*, the mobility models supports basic Car-Following Model (CFM) and Lane-Changing Model (LCM) [95–97]. More vehicular mobility models may be included in the future release.

On the other hand, APP layer processes different types of messages based on different mobility models. Within the scope of this paper, WAVE Short Messages (WSMs) is used. WSMs may vary depending on the applications. For instance, one type of WSMs with moderate priority named Basic Safety Messages (BSMs) broadcasts 10 times per second.

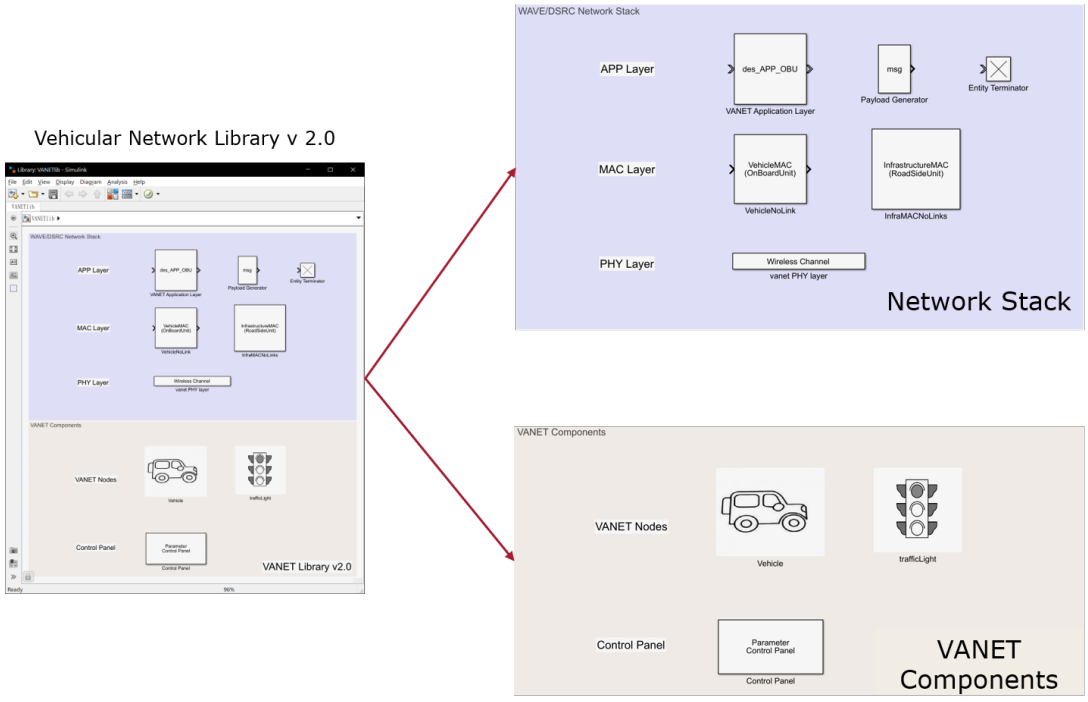


Figure 3.9: Vehicular Ad-hoc Network (VANET) Library.

MAC layer in vehicular network is defined in Wireless Access Vehicular in Environment (WAVE) protocols, among which IEEE 1609.4 explicitly points out Enhanced Distribution Channel Access should be used for the purpose of Quality of Service (QoS). EDCA classifies

all messages in the same channel into 4 priorities, messages with different priority have different channel access deference period. The design purpose of EDCA is to let the message with the higher priority gain channel access more frequently.

A vehicular network includes Vehicle-to-Vehicle (V2V) communication and Vehicle-to-Infrastructure (V2I) communication. In V2V communication, the On-Board Unit (OBU) directly communicates with OBUs from other vehicles. In V2I communication, OBUs communicate with Road-Side Units (RSUs) installed in the static infrastructure along the road. As shown in Figure 3.9, two MAC blocks are created to support OBU and RSU in separate.

PHY layer in vehicular networks is defined in Dedicated Short Range Communication (DSRC) standard. The wireless channel is derived from IEEE 802.11a protocol working on 5GHz with 10MHz channel bandwidth. Roughly saying, the wireless channel of vehicular networks are based on Wi-Fi and it is such a mature technology that a MATLAB toolbox called *WLAN System Toolbox* fully supports all features of PHY layer.

Users can compose vehicular network nodes including vehicles and traffic infrastructures, by dragging and connecting the required blocks from vehicular network stack together. The lower section (yellow) in Figure3.9 contains several created vehicular networks nodes. For example, the vehicle block is made by both APP layer block and MAC layer block. Users can choose the finished blocks to start a quick simulation or create their own blocks according to their requirements.

VANET Toolbox supports two way to run the simulation, *Run from Simulink* shown in Figure3.10 (left) and *Run from MATLAB UI* , Figure3.10 (right):

1. *Run from Simulink*: Users can create an empty Simulink model and drag the blocks from *VANET Library* directly to the empty model. After configuring the necessary parameters, the model is good to run. The advantage of this option is that configuring and tunning parameters are very easy. Users just need to double click the target block and tweak the parameters. The disadvantage is that user may be hard to create a large scale simulations. Imagine a simulation with 100 cars, it is impractical to drag 100 cars from the library and configure each one of them one by one. Thus this method is suitable to build a simple and preliminary model for testing purpose.

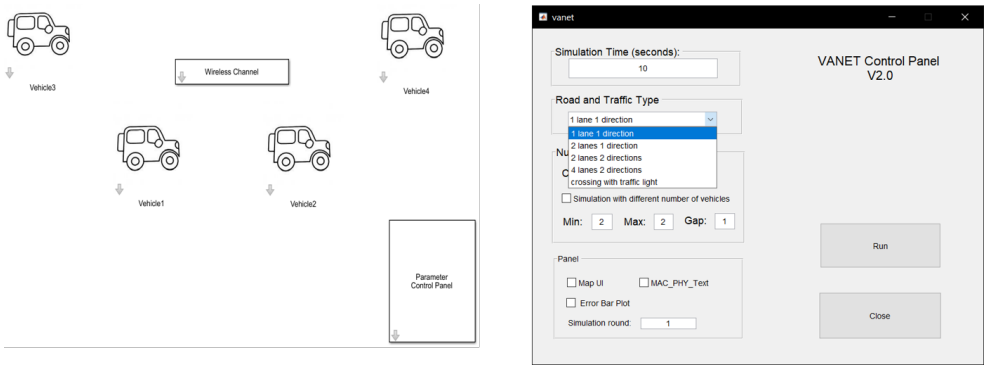


Figure 3.10: Vehicular Network Simulator Mode: Script Panel and Simulink UI.

2. *Run from MATLAB UI*: *VANET Toolbox* is equipped with a VANET Control Panel UI, by which users can control the necessary parameters including simulation time, number of vehicles and traffic model *etc.* It is suitable for large scale simulations. However, a small control panel UI cannot fit all parameters in a vehicular network model. If users need to tweak some parameter the control panel UI does not support, this option is not suitable.

There is one way to compensate the drawbacks of the above two options, *i.e.,* running from MATLAB script. More details will be introduced in the later sections. In this chapter, we will introduce the implementation of *VANET Toolbox* using MATAB Discrete-Event System (DES). The performance analysis of V2V models and the limitations will be introduced in Chapter 4.

3.2.1 Modeling the PHY Link in MATLAB DES

Figure 3.11 shows the DES of wireless channel in PHY layer. The Tx and Rx of PHY layer are integrated with MAC layer. As the PHY wireless channel contains only one data type (waveform), thus only one set of storages, entities and actions are involved. In the figure, a waveform enters the PHY wireless link block from the input port and is stored in the storage. This behavior triggers the entry action *waveformEntry( )*, in which a timer event is called. This time delay is to simulate the air propagation of a waveform.

After the delay period, the corresponding action *waveformTimer( )* is triggered, which

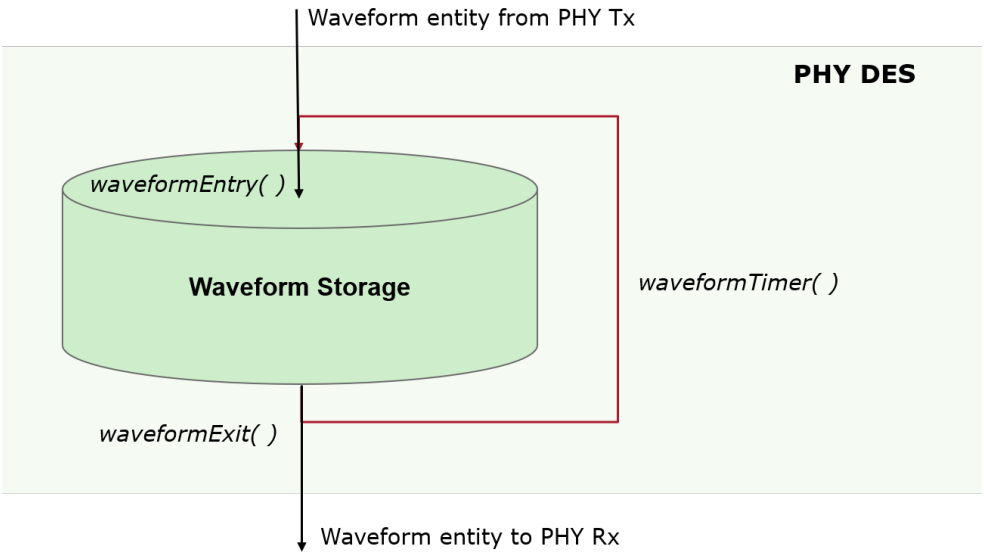


Figure 3.11: Modeling the PHY link using MATLAB DES. The waveform entity enters the DES module, stays for a period and then left the module. Only one waveform type storage is created. waveformEntry(), waveformTimer() and waveformExist() are actions.

stores the waveform in a persistent variable called *waveformBuff* in order to simulate packet collision situation. If more than one waveforms are sent into the PHY channel, they will all be stored in *waveformBuff*. These waveforms may overlap to each other and generate errors.

Then a new waveform is extracted from the persistent variable, no matter if corrupted or not, it is sent to the output port via event *obj.eventForward( )*. Once the waveform left the wireless channel DES, action *waveformExit( )* is called to reset the channel persistent variables.

3.2.2 Modeling the MAC Layer using MATLAB DES

Figure 3.12 illustrates the design of a MAC DES module, in which one payload type storage, six frame type storages, and one waveform type storage are defined to contain payload, frame, and waveform entities, respectively. The *payload type* entity enters into the MAC layer from the APP layer and stays in the payload storage. In the corresponding *payloadEntry() action*, a new event *frameGenerate()* is called, which converts the payload

entity to a frame entity by adding the necessary header and trailer.

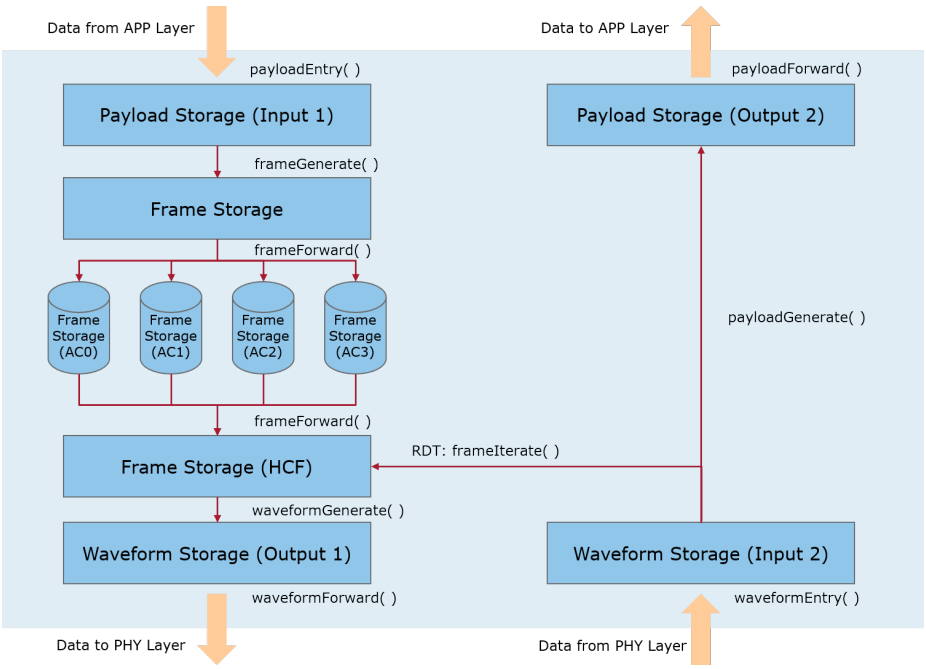


Figure 3.12: Modeling the MAC layer using MATLAB DES. The MAC DES module involves threes types of entities: payload entity, frame entity and waveform entity. It is responsible for the data streams sending to the PHY layer and receiving from the PHY layer.

Based on the priority, the frames are forwarded into 4 AC queues via *frameForward( )* event. In AC queues, frames with different priorities experience different channel access deference (AIFS+backoff). Once the backoff period is done, the frame is forwarded to *Frame Storage (HCF)*. It is possible that more than one frames are ready to send, this will cause internal contention. When this situation happens, only the frame with the highest priority will be forwarded to the HCF frame storage, others remain in the original AC storage and redo the backoff.

The winner frame checks the external contention status and if the channel is idle, the frame is converted to a waveform via action *waveformGenerate( )*. If the channel is busy, the frame stays in the HCF storage until the channel becomes idle. A waveform is converted from a frame using PHY Tx mechanism mentioned in the above section. Then the waveform is forwarded to output 1 by waveform forward event.

In a reliable data transmission(RDT), a unicast waveform is required to have an ACK

returned. After the RDT waveform entity left the MAC DES, a timer event is attached to the original frame entity in HCF storage. If the ACK is not received within the timer period, this frame entity will be converted to the waveform entity and forwarded out again. If the ACK is still not received after the maximum retransmission limit is reached, this frame is destroyed to prevent further retransmission. If the needed ACK arrives in time, an iteration event is called. In the corresponding action *frameIterate ( )*, the frame in HCF storage is destroyed, and new frames start to contend for channel access.

It is worth mentioning that the reason why the timer is attached to the frame entity in HCF storage instead of the waveform entity is because after the waveform entity left the MAC DES, it no longer exists in the object. All unfinished events attached become invalid immediately and will never happen. The waveform entity is generated based on the corresponding frame entity, after the waveform entity left, the frame entity stays in the storage in order to keep the attached events valid.

When an waveform entity enters the MAC DES via input port 2, it stays in the connected waveform storage. This process triggers the action *waveformEntry( )*, in which the type of waveform will be checked. If it is an ACK for RDT, an iteration event will be called. In the corresponding action *frameIterate( )*, the target frame will be destroyed and its unfinished events are all becoming invalid. If the waveform entity is a data type waveform, the data information is extracted by waveform entry action and a new payload is generated containing the extracted data. The payload is finally forwarded to the upper layer via output port 2.

3.2.3 Modeling the APP Layer using MATLAB DES

APP layer is the top layer and responsible for generating the first entity to trigger the whole discrete event system. Instead of receiving entities from external entity generator blocks, a better option is generating entities inside APP block. This can be implemented by *events=setupEvents(obj)* methods from class *MATLAB.DiscreteEventSystem*, which sets up the first set of entity generation events at the start of simulation. The code is listed below:

```
function events=setupEvents(obj)
events=[obj.eventGenerate(3,'driving',0.1,300), ...
obj.eventGenerate(1,'BSMgen',0.1,100);];
end
```

The above code generates two entities. The first entity is generated in storage 3 with tag of 'driving'. The entity stays in storage 3 and serves as the seed to trigger a series of driving behaviors. After the 'driving' entity is generated, a 0.1 second timer event is attached to the entity. Once the delay period is ended, another 0.1 second timer event is triggered.

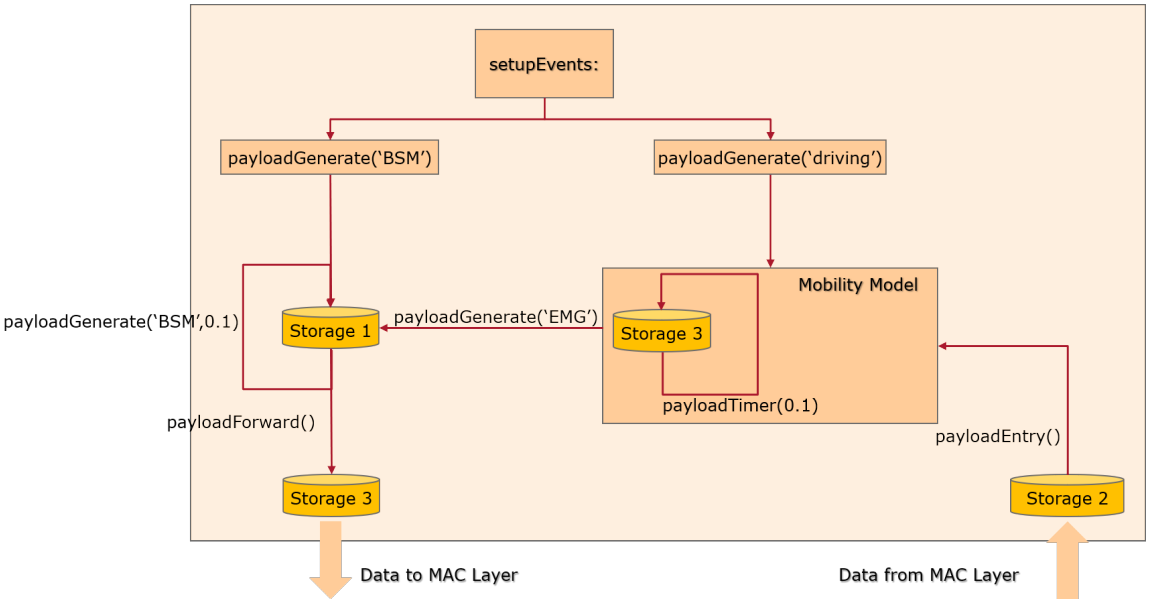


Figure 3.13: APP Layer Design using MATLAB DES.

The vehicular mobility models are integrated with the APP layer. This means that the APP DES module has two responsibilities. First, the APP DES module should generate payload entities containing traffic information and sent to the MAC layer. As shown in Figure 3.13, two payload storages resource (storage 1 and storage 3) are involved, while the *setupEvents()* generates Basic Safety Messages (BSMs). The 0.1 in *payloadGenerate('BSM',0.1)* event indicates the payload generation intervals since BSMs are generated at a rate of 10 Hz. Once generated, the payload entities are forwarded to Storage 3 and

finally sent to the MAC DES Module.

The timer event and timer action are mutually triggered in a loop with 0.1 second gaps. During the 0.1 second period, each vehicle updates its location based on the mobility model. Meanwhile, each vehicle inquires the 'global database' to see if it is involved in an car accident. Thus every 1 second, each vehicle updates its position 10 times and conducts car accidents inquiries 10 times. More information about the 'global database' will be introduced in the next section.

The second entity is generated in storage 1 containing vehicle's driving information including speed, position *etc.* The information is included in a Basic Safety Message (BSM). Each vehicle is required to broadcast BSM 10 times per second. Therefore, every 0.1 second, an entity generate event is triggered to create a BSM. The BSM is forwarded to storage 3 and finally left the APP block via output port.

When receiving payloads from the MAC DES module, the APP DES module extracts the message from the payload entities and forwards to the mobility models. The traffic information will be updated according to the received message. A Emergency (EMG) type message can be generated upon request by the vehicles.

The APP DES module is also responsible for making the vehicles to move on the road according to mobility models, thus a new type of entity with a *driving* tag is created and stored in Storage 3. This *driving* entity stays inside Storage 3 forever and will never be sent out. A timer event is recursively triggered on the *driving* entity at an interval of 0.1 seconds. This is the refresh rate of the vehicles moving across the map. The corresponding timer action *payloadTimer(driving)* is called every 0.1 seconds to update the driving information including vehicle position, speed, and direction. This refresh rate can be increased at the cost of execution speed.

3.2.4 Peripheral Function Implementations

Build Isolated Vehicle Block and Connectionless Links

We have built the necessary layers, *i.e.,* APP layer, MAC layer and PHY layer, for a vehicular network node. A vehicle node now has the capability to communicate with other

vehicles. Before performing a simulation, more factors need to be considered. *'a simple example'* in Chapter 2 shows that Simulink blocks are connected to each other via 'lines'. These lines may be feasible if the scale of a Simulink model is small or medium. For a large scale simulation, drawing all lines one by one becomes a severe limitation.

The simulation for vehicular networks is often in a large scale. For example we want to simulate V2V communications among 100 vehicles. Suppose all V2V nodes are directly connected via independent links, the maximum number of lines needed is $\frac{100\times(100-1)}{2} = 4950$. Imagine how tedious to draw 4950 lines. Thus it is necessary to get rid of these lines.

The first option is using 'From' and 'Goto' tags from Simulink library. Users can set different tag to 'From' and 'Goto' pairs. Any pair with the same tag are actually connected via invisible lines. In order to create 'From' and 'Goto' pairs in batches, we convert the MAC DES block into a subsystem and seal it with mask. In the 'Initialization commands' tab of the 'Mask Editor', run the predefined mask codes to generate 'From' and 'Goto' tags in batches.

Figure 3.14 show an example of 'From' and 'Goto' pairs on vehicle 12. The naming rule of 'From' is defined as 'receive(r) + fromVehicleID + toVehicleID'. For example, 'r0112' indicates that the link is from vehicle 1 to vehicle 12. Similarly, the 'Goto' tag is defined as 'to (t)+fromVehicleID + toVehicleID'. For instance, 't1201' means the link starts from vehicle 12 and invisibly connects to a 'Goto' block attached to vehicle 1 with a tag of 't1201'.

Suppose all vehicles are connected to each other via invisible links, and each vehicle is indexed by numerical ID. Based on the total number of vehicles, the ID of vehicles on both end of each link can be calculated using the following code:

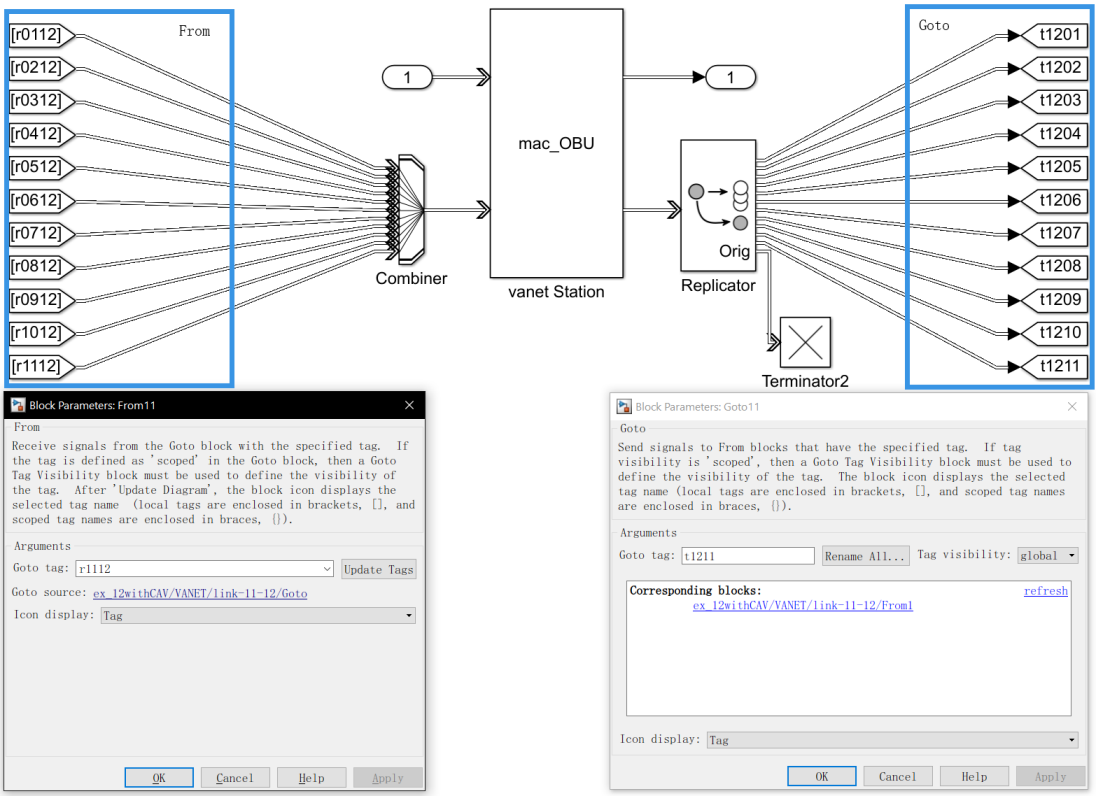


Figure 3.14: Examples of 'From' and 'Goto' pairs created in batches by MATLAB code.

```
function [LinkFrom,LinkTo]=fcn_GetLinks(numStations)
for i=1:numStations-1
for j=2:numStations
if j>i
LinkFrom=[LinkFrom i];
LinkTo=[LinkTo j];
end
end
end
end
```

Then we calculate the number of 'From' and 'Goto' pairs according to the links. The

code below automatically create all the 'From' and 'Goto' pairs we need.

```
num = length(Conns);
set_param([block '/Combiner'], 'NumberInputPorts', num2str(num));
set_param([block '/Replicator'], 'NumberReplicas', num2str(num));
for i = 1:num
bpath = [block '/From' num2str(i)];
add_block('built-in/From', bpath);
set_param(bpath, 'GotoTag', ['r' src dst]);
add_line(block, ['From' num2str(i) '/1'], ['Combiner/' i]);
bpath = [block '/Goto' num2str(i)];
add_block('built-in/Goto', bpath);
set_param(bpath, 'GotoTag', ['t' dst src]);
add_line(block, ['Replicator/' num2str(i)], ['Goto' i '/1']);
end
```

Even though the vehicle blocks are not constrained by the lines, users still have to create the same amount of 'From' and 'Goto' pairs. Actually the lines between 'From' and 'Goto' remain the same but invisible, see Figure 3.15 . In the figure, three nodes of a network are communicating via $\frac{3\times(3-1)}{2} = 3$ links. Each link contains a pair of wireless channel link (one for sending and one for receiving) as well as two pairs of 'From/Goto' blocks. Suppose we have n vehicle nodes in a network, the total number of wireless channel links is $\frac{2\times n\times(n-1)}{2} = n(n-1)$ and the total number of 'From/Goto' blocks is $\frac{4\times n\times(n-1)}{2} = 2n(n-1)$. This feature not only increases the configuration difficulty, also slow down the simulation speed tremendously. The only advantage of this design is the model can simulate *Hidden Terminal Problem* by manipulating each links.

Since MATLAB 2017a, *VANET Toolbox* involved into the $2nd$ version by replacing the 'From/Goto' pair with 'Multicast' blocks. 'Entity Multicast' and 'Multicast Receive Queue' are blocks from SimEvents Toolbox. Unlike the 'From/Goto' pair, which only supports one-to-one entity transmission, multicast blocks support multiple-to-one or one-to-multiple transmissions. Besides, the tag pair in 'From' and 'Goto' has to be unique in a model,

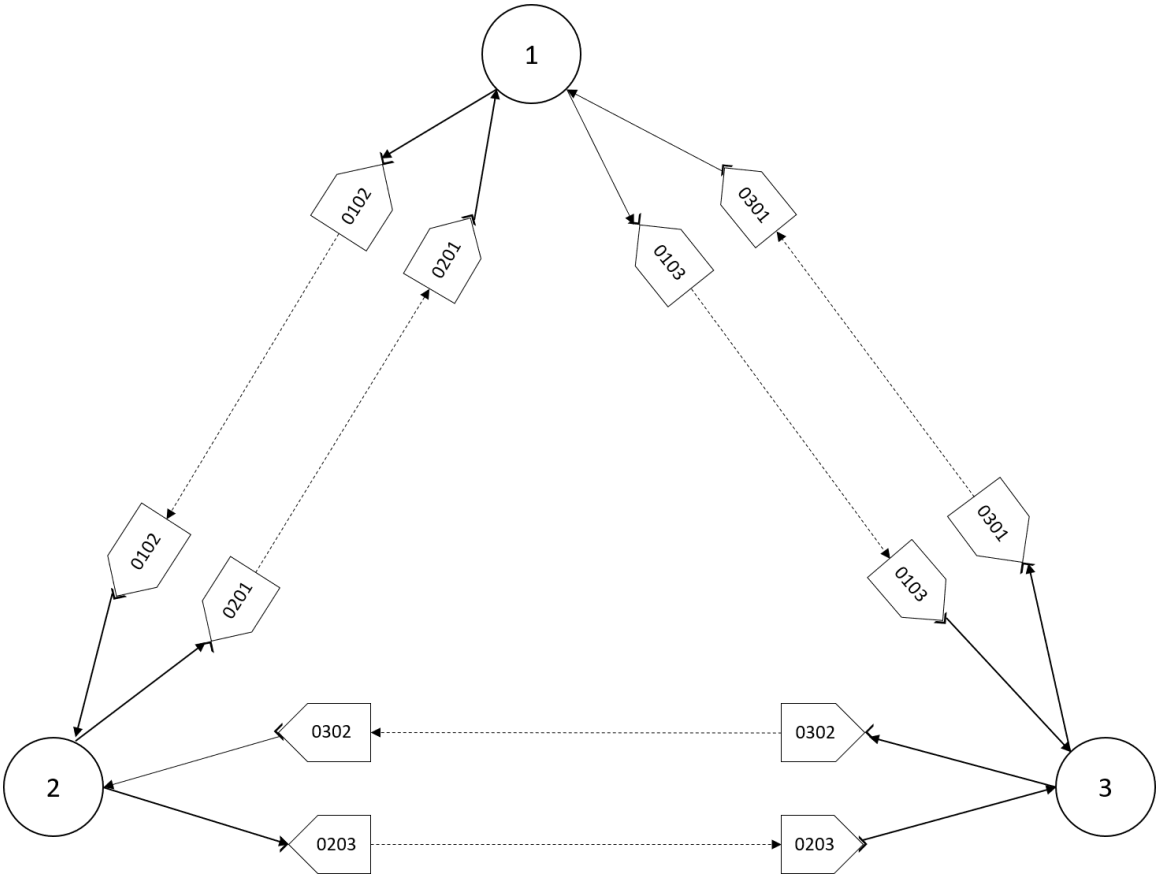


Figure 3.15: Illustration of three nodes with unicast tag pairs.

i.e., only one pair of 'From/Goto' with same tags is permitted in the same model, while multicast blocks allow more than one tags with the same names.

Figure 3.16 shows an example of a wireless link with multicast blocks. In the figure, each vehicle is attached with an 'Entity Multicast' block with the same tag name of 'toPhy'. All the waveform entities sent from different vehicles are received by the 'Multicast Receive Queue' attached to the wireless link with the same tag of 'toPhy'. After processing by the wireless link DES, the waveform entities are sent out via another 'Entity Multicast' block with tag of 'fromPhy'. Each vehicle will receive a copy of waveform entities from its own 'Multicast Receive Queue' with tag of 'fromPhy'. Based on the Rx checking mechanisms we described in the above sections, MAC layer DES of each vehicle will keep the waveform entity right to it and destroy all the other irrelevant waveform entities.

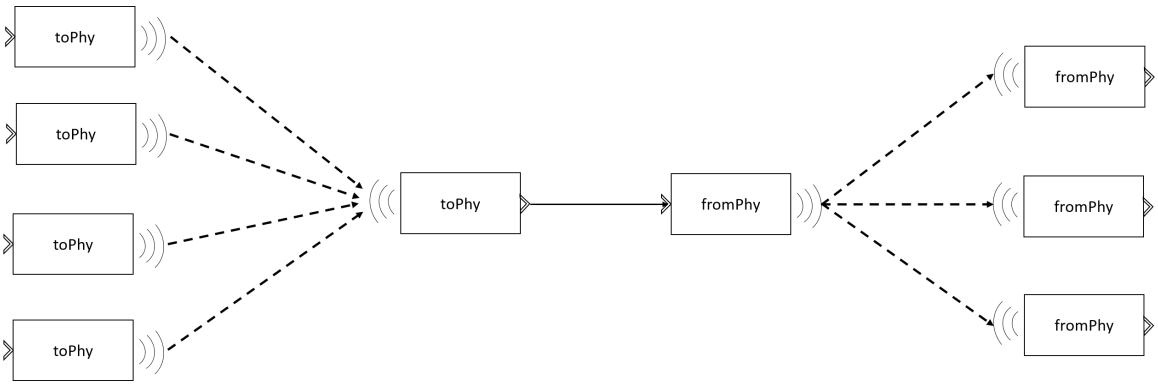


Figure 3.16: Illustration of three nodes with multicast tag pairs.

In this mechanism, the wireless link blocks decreases to 2 in a model (bidirectional communication), and no matter how many vehicle nodes in total, the PHY links are alway 2. For n vehicle nodes, the number of multicast pairs is $2n$. Compared with unicast 'From/Goto' pair option, multicast pair greatly decreases the design complexity and increase the simulation speed.

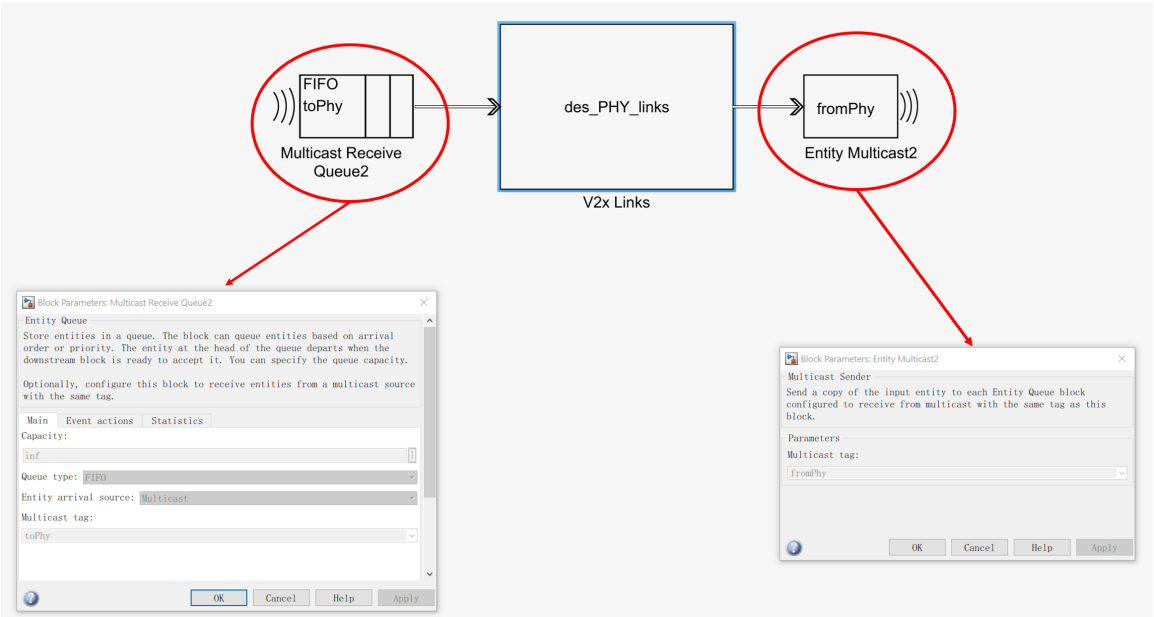


Figure 3.17: Configurations of multicast blocks.

Figure 3.17 shows the mulitcast block pair and their configuration parameters. In the

figure, all data are received by 'Multicast Receive Queue' with tag of 'toPhy'. The 'Multicast Receive Queue' is set to be a FIFO queue with infinite capacity. After processing by the wireless channel links, *des_PHY_links*, the data are broadcast via 'Entity Multicast' block with tag of 'fromPhy'. In this way, all 'Multicast Receive Queue' blocks attached to each vehicle node with tag of 'fromPhy' will receive a copy of data.

Local Database and Global Database

In a vehicular network, vehicles keep sharing driving and traffic information to each other. These information guide vehicles on making decisions to their driving behaviors. In *VANET Toolbox* simulator, each vehicle is a set of objects, *i.e.,* a combination of APP, MAC and PHY object. Here in order to describe more conveniently, we treat each vehicle as a single object that stores the traffic information.

The data structure MATLAB supports includes array, struct and containers.Map. *Array* in MATLAB can only store numerical values, *i.e.,*numbers. The data structure we are looking for should act more like a database. To this point, array is not a good option to perform database type operations such as adding, removing and information inquiry. *Struct* supports 'key-value' pairs but its inflexible on data type makes it not unfeasible to be a database. *Containers.Map* is the best option because it can save any type of data in 'cells'. It supports 'key-value' pair index and data inquiry. What's more, the data in a container.Map are save as hash-map, which increases the read/write speed. Thus we select containers.Map as the database to store all the traffic information.

The next challenge is how to maintain the traffic information during the simulation process. MATLAB provides two ways to make this happen, properties in an object and persistent variables.

1. Ojbect properties: attaching the database (containers.Map) to a private property inside an object is a good option. The value of properties remains valid during a simulation. And it is convenient to query the needed information based on class name and property names. However, as of MATLAB 2018a, SimEvents toolbox does not support containers.Map inside objects when using *Code Generation* mode. Code

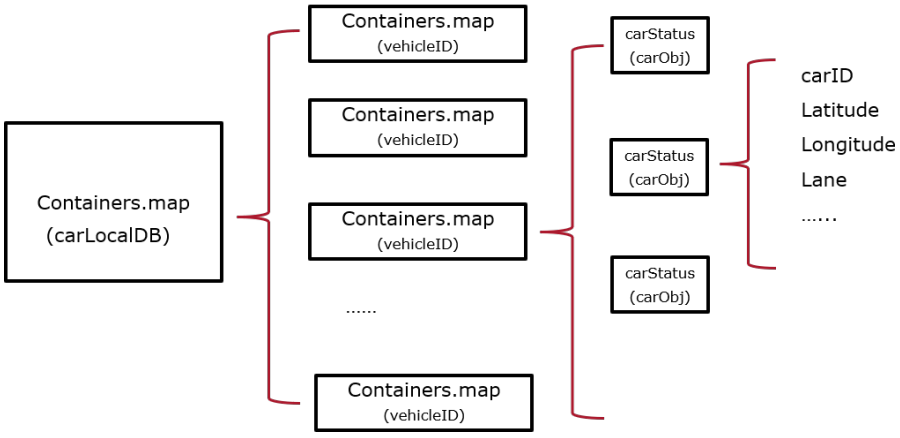


Figure 3.18: Design of Local Database.

generation is a feature that the MATLAB code can be converted to C/C++ code in order to increase the simulation speed. According to this limitation of code generation, it is regretful to gave up this option.

2. Persistent variables: persistent variables in MATLAB are like static variables in C/C++, which maintains values during the simulation. The database (containers.Map) is attached to a persistent variable. The operations on this persistent variable are coded in a MATLAB function file. Note that persistent variables are not supported by code generation either. Therefore, when using functions containing persistent variables inside a DES object, we need to *coder.extrinsic(functions)* in order to bypass the code generation. This is a compromise between code generation and persistent variables, and this compromise will surely slow down the simulation speed. But using persistent variables in a function and *coder.extrinsic* the function inside the object to bypass code generation is the only option.

In a vehicular network, each vehicle has its own traffic information, thus each vehicle should have its own local database. The local database is called 'carlocalDB' in *VANET Toolbox*. However, as mentioned above, the persistent variables do not support code generation and will slow down the simulation speed. Therefore, the design of databases should contain as few persistent variables as possible. The challenge here is how to use a *single* persistent variable to contain all carLocalDBs separately for each vehicles. Figure 3.18

shows the concept of design to local databases. *carLocalDB* is designed as two-level cascading containers.Map. The top level containers.Map is *carLocalDB* attached to the persistent variable *localDatabase*. The second level containers.Map is for each vehicle indexed by vehicleID. Each second level containers.Map contains *carStatus* object to save and maintain the traffic information. For instance, upon receiving a BSM, a vehicle registers its traffic information to *carLocalDB*, partial of codes are shown below:

```
if ~isKey(localDatabase,vehicleID)
localDatabase(vehicleID)=containers.Map('KeyType','double', '
    ↪ ValueType','any');
end
localDB=localDatabase(vehicleID);
if ~isKey(localDB,vehicleID)
localDB(vehicleID)=carStatus;
end
carObj=localDB(vehicleID);
carObj.carID=vehicleID;
```

While driving, a vehicle keeps checking its *carLocalDB* to calculate its distance to the other surrounding vehicles. If the distance to the other vehicles is shorter than a pre-defiend safety distance, an action may be conducted such as braking, lane changing *etc.* Partial code on inquire *carLocalDB* is listed below:

```
if isKey(localDatabase,vehicleID)
localDB=localDatabase(vehicleID);
longitude=localDB(vehicleID).longitude;
latitude=localDB(vehicleID).latitude;
...
index=localDB.keys;
end
```

In order to test the performance of vehicular networks, all vehicles are fully autonomous

and communicating with each other with only V2V communication. In other word, all vehicles are blind without wireless communication. This may result a weird situation, that is if the network load is heavy, BSMs may be delayed or corrupted due to packet collisions. Thus vehicles may not be able to receive all BSMs correctly on time and vehicle collisions may happen.

Under normal circumstance, when a car accident happens, both cars should stop for a while to process the accident. In a vehicular simulation created by *VANET Toolbox*, the situation may be different. Suppose vehicle 2 hits vehicle 1 due to the lost of BSMs, under current design, vehicle 2 has no idea that it have caused a car accident and will keep driving. On the simulation GUI, users may observe that vehicle 2 drives 'through' vehicle 1 without stopping. This obviously disobey the nature rules. To prevent such situation, we need to create a global database named *carGlobalDB* shown in Figure 3.19, which records all vehicles' actual position information and the recording of these information is not affected by the network communication quality.

carGlobalDB should start to track the driving information of each vehicle, including carID (vehicleID), position, speed, at the beginning of a simulation. *carGlobalDB* should not interfere the driving pattern unless car accidents happened. Compare with *carLocalDB*, *carGlobalDB* is relatively simple, because all vehicles are inquiring the same copy of data and the data are not classified by different vehicleID.

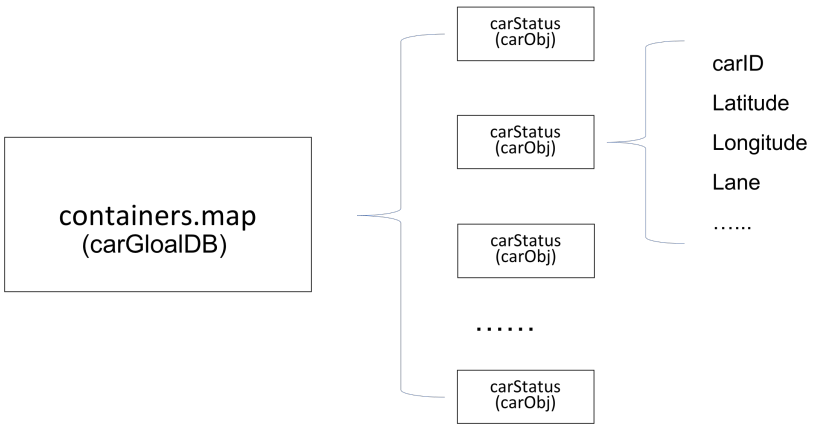


Figure 3.19: Design of Global Database.

In Figure 3.19, we create a containers.Map and attach it to a *persistent* variable. The 'keys' are carID and the 'values' are carStatus object. All vehicles register their traffic information every 0.1 second, the code is shown below:

```
globalDatabase(vehicleID)=carStatus;
% Create carStatus object to containers.Map
carObj=globalDatabase(vehicleID); % Select the carStatus object
carObj.carID=vehicleID; % Save values
carObj.latitude=curPositionX;
carObj.longitude=curPositionY;
carObj.lane=lane;
```

While saving traffic information to *carGlobalDB*, each vehicle inquires the database at the same time and check if it is involved into a car accident. Partial codes are listed below:

```
index=globalDatabase.keys;
tempArray=zeros(1,globalDatabase.Count);
inputCarStatus = globalDatabase(vehicleID);
for i=1:globalDatabase.Count
currentCarStatus = globalDatabase(index{i});
end
```

The differences between carLocalDB and carGlobalDB are listed as follows:

1. carLocalDB stores data from each vehicle point of view. As the received messages may be affected by channel imperfection, the information inside carLocalDB may be outdated, inaccurate or incomplete. While carGlobalDB only saves the latest accurate information for *all* vehicles in the simulation.

2. each vehicle has its own carLocalDB and each carLocalDB can be only accessed by its owner vehicle. While there is only one unique carGlobalDB, and all vehicles have the right to access to it.

3. The information from carLocalDB may affect the driving path of a vehicle while the

information from carGlobalDB will not change a vehicle's driving trail unless car accidents are detected.

3.3 Run Simulations using VANET Toolbox

There are three ways to run simulations using *VANET Toolbox*, run from Simulink, run from MATLAB and run from VANET UI, each of which has its benefits and limitations. We will introduce all of them one by one in the following sections.

3.3.1 Run from Simulink

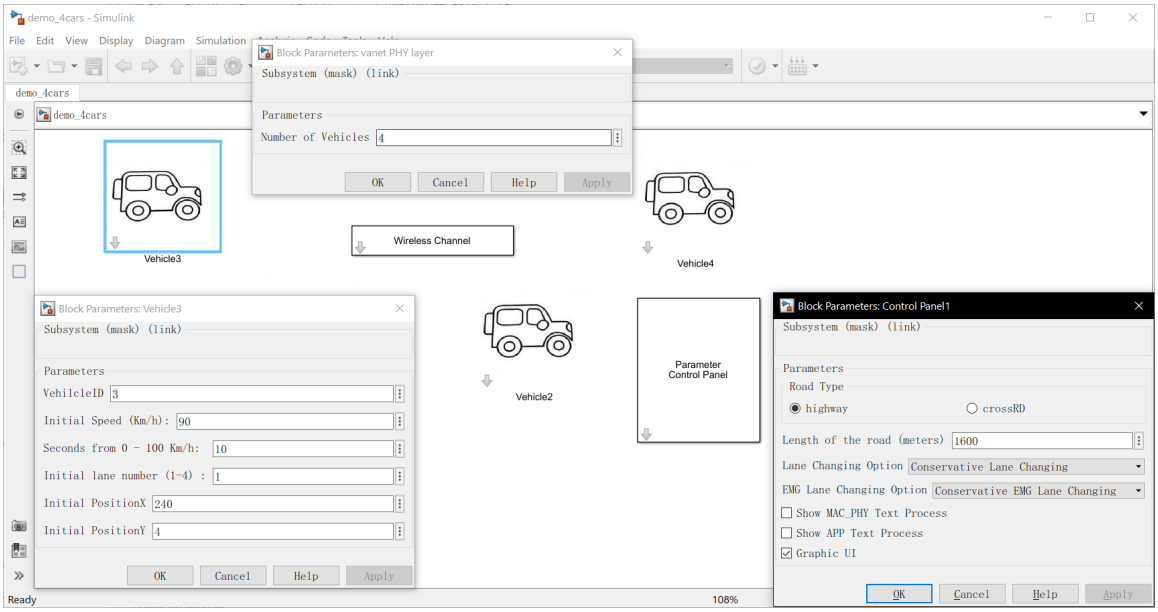


Figure 3.20: V2V Model with 4 Vehicles.

As of R2018a, SimEvents toolbox cannot run without Simulink. VANET Toolbox is developed by MATLAB Discrete-Event System from SimEvents, thus a model created by VANET Toolbox requires Simulink as the simulation environment. Create a Simulink model is the most intuitive way. Figure 3.20 shows a V2V simulation model with four vehicles. Three types of blocks are involved, Vehicle, Wireless Channel and Control panel. Vehicle block is an abstract of a real vehicle. It contains an OBU, which supports APP, MAC

and PHY layer of vehicular networks. The Tx and Rx of PHY layer are integrated in MAC DES, therefore Wireless Channel block only serves as a wireless DSRC channel with two-ray ground reflection model. Control Panel block is responsible for tunning simulation related parameters including road type, road length, lane changing options and text/UI output switch. The details of each block are showing next.

Vehicle Block has the following parameters:

- VehicleID: A traffic model often involves more than one vehicles, each vehicle is identified by a unique id, *i.e.,* VehicleID. VehicleID is initialized at the start of simulation. All traffic information during the simulation including speed, acceleration, position are all classified by vehicleID.

- Initial Speed (Km/h): Users can set a initial speed when a simulation starts. The initial speed should be a positive scaler within the range from 0 to the speed limit.

- Seconds from 0-100 Km/h: Acceleration is evaluated by the time when a vehicle speeds up from 0 to 100 Km/h. A typical acceleration value should range from 5 to 10 Km/h^2.

- Initial lane number (1-4): In the current version, VANET Toolbox supports traffic up to 4 lanes. Users can choose the initial lane from this parameter.

- Initial PositionX: initiate the start point in latitude for a vehicle on the road.

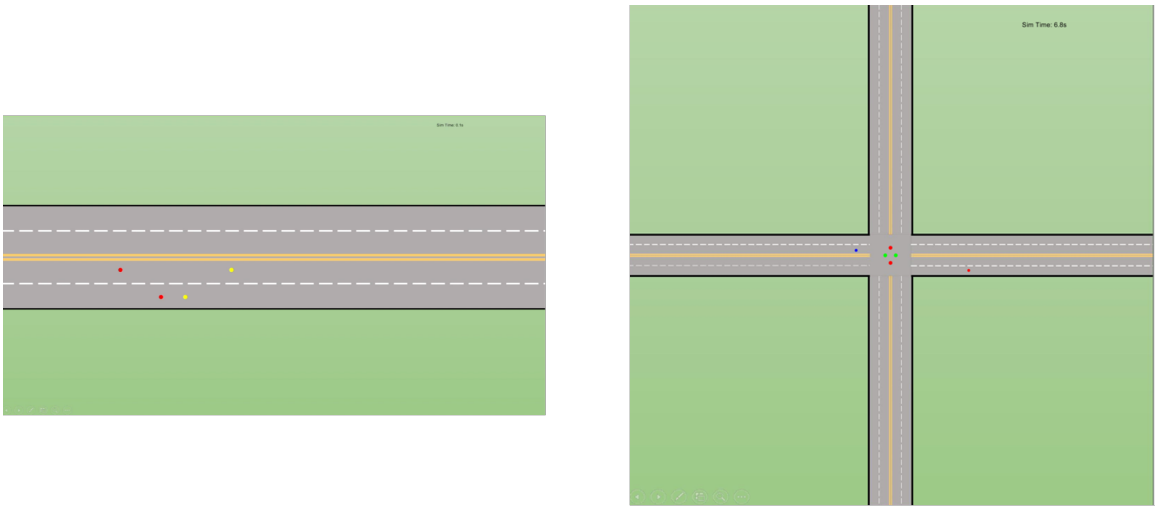
- Initial PositionY: one lane contains two sub-lanes, slow lane and fast lane. PositionY decides which sub-lane a vehicle starts to drive.

The *Wireless Chanel* block contains only one parameter, Number of Vehicles. Users need to set the total number of vehicles before a simulation starts. The number of vehicles is not necessary to compose a simulation model but it is necessary for other simulation control functions to initialize variable or states.

Control Panel block has the following parameters:

- Road Type: *VANET Toolbox* supports highway road model and intersection with traffic light model. The default road type is highway.

- Road length (meters): The total length of road is defined here.
- (EMG) Lane changing option: Lane change may happen under normal condition or emergency condition. Users can choose from conservative lane changing scheme (conLC) , performance lane changing scheme (perLC) or braking only scheme (nonLC). The performance of these three schemes will be evaluated in Chapter 4.
- Outputs and UI: A model created by *VANET Toolbox* can output the simulation process in text format. The text is controlled by APP, MAC layer in separate. The text output may slow down the simulation speed, thus it is often turned on only for debugging purpose. VANET Toolbox also supports graphic user interface (GUI) as shown in Figure 3.21. GUI can intuitively show the results, but it will also slow down the simulation speed. For large scale of simulations, GUI is suggested to be turned off.



A. Highway Traffic

B. Intersection with Traffic Light

Figure 3.21: GUI VANET Toolbox: Highway and Intersection with Traffic Light.

Creating a vehicular network model in Simulink is intuitive and relatively simple. Users can easily tune the parameters described above in each block. However, when the simulator becomes large scale, it is impractical to tune all blocks. VANET UI Panel provides a platform to run simulation in batches.

3.3.2 Run from VANET UI Panel

VANET Toolbox provides a control UI to create and run simulation models, as shown in Figure 3.22. The major parameters are included in the panel such as number of vehicles, simulation time *etc.* One thing needs to mention is the 'Min-Gap-Max' set, which allows users to repeat simulations with different number of vehicles from 'Min' to 'Max' with step increment of 'Gap'. Suppose $Min : 4, Gap : 2, Max : 10$, the simulation starts with 4 vehicles, when it's done, a new simulation with 6 vehicles is started, then 8 vehicles and 10 vehicles.

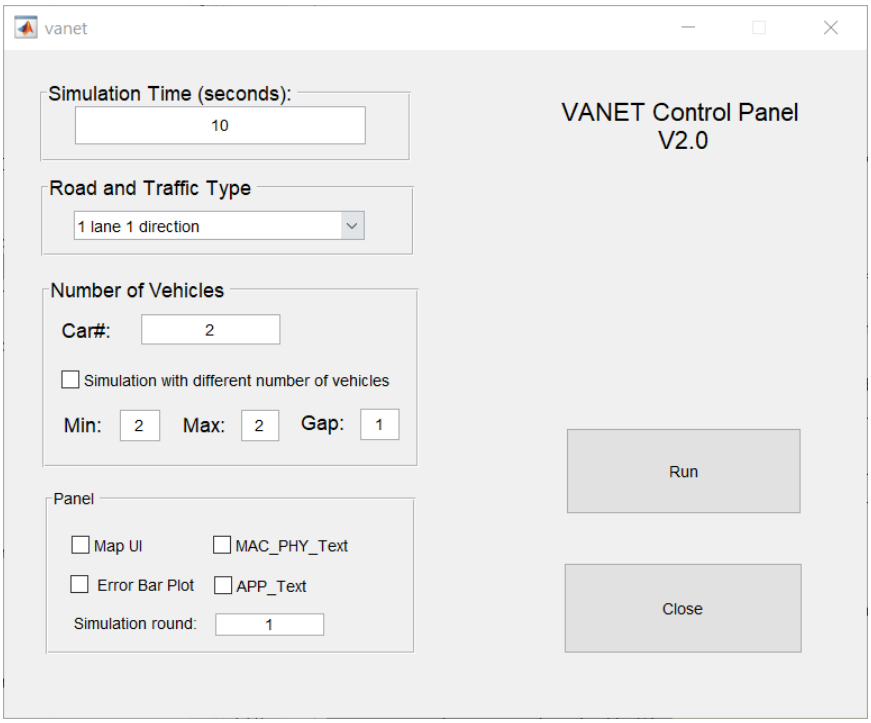


Figure 3.22: VANET Control Panel to Create and Run Simulations.

The control UI is suitable for repeating simulations with necessary parameters only. The interface is designed by a MATLAB tool named *GUI Design* (GUIDE). Users can open the design panel by typing 'guide' in the command windows. Figure 3.23 (left) shows all type of GUI templates it supports. We choose a default template as an example, shown in Figure 3.23 (right). On the design area, we create a series of buttons, text field or tables. In the

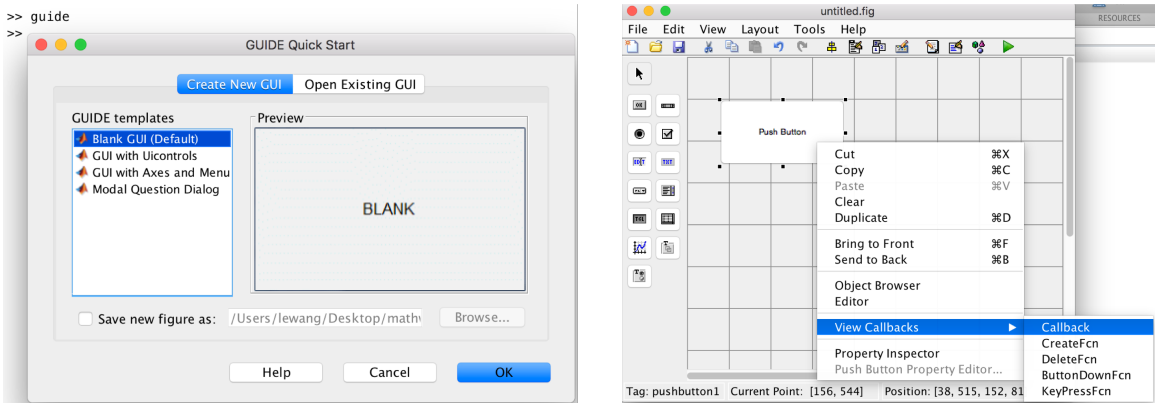


Figure 3.23: GUI Design Panel and An GUI Example.

figure, we created a push button. It is just an empty button, which currently does nothing until we create the callback function of it.

In next section, we will show you how to create a model and run a simulation via MATLAB functions. The MATLAB function has some input parameters such as simulation time *etc.* Users input the parameters to the UI panel, then all these parameters are collected beneath the UI panel interface and finally sent to the MATLAB function as inputs. In this way, the UI panel is successfully connected with the MATLAB function.

The value for each button or text object on the UI panel can be obtained by each handlers. For example we can get the value of simulation time (simTime) using the following code *simTime=str2num(get(handles.simTime,'String'))*, in which 'simTime' is the name of the text field, the values users input is contained in 'handles.simTime' and the data type is 'string'. We obtain this string type value using 'get'. In MATLAB functions, to run simulations, simTime is in double type, thus the string type simTime is converted by 'str2num' command. The MATLAB function is associated with the *runButton* method because the simulation is started by clicking the 'Run' button on UI Control Panel. Partial code of this button is listed below:

```
simTime=str2num(get(handles.simTime,'String'));
roadTypeOpt=get(handles.roadType,'String');
roadTypeIdx=get(handles.roadType,'Value');
roadType=char(roadTypeOpt(roadTypeIdx,:));
multiNumVehicles=get(handles.multiVehilcleNum,'Value');
mapUI=get(handles.mapUI,'Value');
simRound=str2double(get(handles.simRound,'String'));
...
runModel(simTime,roadtype,minVehicleNum,maxVehicleNum,gap,simRound,
    ↪ errBar,macTXT,appTXT,mapUI);
```

In the code, we collect parameters including simulation time (simTime), road type (roadType), number of vehicles (multiNumVehicles), number of repeating simulations (simRound) and if showning a MAP UI (mapUI) from each handlers. These parameters are passed to *runModel* as inputs. When then 'Run' button on Control UI Panel is clicked, function *runModel* starts to create a model based on the input parameters and run the simulation(s).

Actually, more parameters can be tunable in addition to the above mentioned selections. It is infeasible to contain all tunable parameters in the currently Control UI Panel. Therefore, this option is constrained by limited parameters.

3.3.3 Run from MATLAB Code

In order to run simulations in large scales and, at the same time, tune all parameters as needed, the only option is running simulations via MALTAB code. Essentially, the MATLAB function does nothing but creating an invisible Simulink model using MATLAB code. The process of copying blocks from library, setting up the simulation paramters is concealed beneath the MATLAB code. But the advantage is tunning up parameters in batches becomes easy. For example, copying 100 vehicle blocks from VANET Library to an empty Simulink model can be implemented by one line of code. Besides, changing the

parameters of these 100 vehicles can be implemented by a *for* loop in MATLAB code, unlike in Simulink, users have to double click each vehicle block to change the parameters and repeat 100 timems.

To run simulations using MATLAB code, we first need to create an empty Simulink model using the following code:

```
h=new_system; % create an empty model
mdl=get_param(h,'Name'); % get the model name
```

Next, we obtain blocks from *VANET Library* and add to the new model *mdl* using MATLAB command *add_block*. The format of *add_block* is 'add_block(source block, destination block, parameters)'. Take a V2V communication model for example, the required blocks include 'Vehicle', 'vanet PHY layer' and 'Control Panel'. The code to add 'Vehicle' is shown below:

```
add_block('VANETlib/Vehicle', [mdl '/Car' num2str(id)], ...
'carID', num2str(id), ...
'startSpd',num2str(car.Speed),...
'startAcc',num2str(car.Acceleration),...
'initLane',num2str(car.lane),...
'startPosX',num2str(car.PositionX),...
'startPosY',num2str(car.PositionY));
```

The code above first obtain *Vehicle* block from the library *VANETlib*, then copy it to *mdl* with name of 'Car'. A vehicular network model usually contains more than one cars, each car is classified by different IDs. The *Vehicle* block has parameters such as carID, startSpd *etc*. These parameters can be tuned in the code or kept with the default values. Similarly, the 'vanet PHY layer' and 'Control Panel' blocks are added to *mdl* in the following code:

```
add_block('VANETlib/Control Panel', [mdl '/controlPanel'], ...
'txtEnable',n.txtEnable,...
'appTXTEnable',n.appTXTEnable,...
'isUIon',isUIon,...
'road','highway',...
'roadLength','1680',...
'laneChangingOption',laneChangingOption);
...
add_block('VANETlib/vanet PHY layer', [mdl '/VANET'], ...
'numStations', num2str(numVehicles));
```

After adding all necessary blocks to the model with configurations, the last step is setting simulation related parameters. This can be done via *set_param* command. Then the model is ready to run. The code is listed below:

```
set_param(h,'StopTime',num2str(simTime)); % Set simulation period.
set_param(mdl,'StopFcn',stopfcn); % Set functions running after the
    ↪ simulation is done.
sim(mdl);        % Start to run simulation
close_system(mdl,0);     % close the model after simulation
```

3.4 Chapter Summary

In this chapter, we have presented an integrated vehicular network simulator, *VANET Toolbox*, developed by MATLAB Discrete Event System (DES). This is the first vehicular network simulator in MATLAB/Simulink environment that supports full stack of network protocols. Design structure of the main components, APP layer, MAC layer, PHY layer and the basic mobility models, were proposed and we have shown that *VANET Toolbox* is capable to simulate the inter-communication between vehicles under different scenarios but is not without limitations. *VANET Toolbox* requires only MATLAB/Simulink background

to use it. The knowledge of the developing tools including *SimEvents*, *Communication and WLAN System Toolbox* are necessary if users want to modify *VANET Toolbox*. We will open-source *VANET Toolbox* along with several examples implementations created by it. The design purpose of *VANET Toolbox* is to provide a framework and opportunity to attract more researchers to improve it and finally benefit the vehicular network development.

Chapter 4

Performance Analysis of Vehicular Network Simulations on V2V

In this section, the performance of the proposed simulation environment is evaluated. We first discuss the computational costs of a full-stack vehicular network simulator in terms of the number of events scheduled during the simulation. Then, we compare the packet success rate (PSR) of BPSK in a AWGN channel between MATLAB PHY layer implementation and NS-3 error rate model. Due to the bit level processing of the PHY layer, the channel tracking (CT) techniques can be enabled on the L-LTF field in order to cope with the high Doppler spread. The performance of CT is evaluated for BPSK signal with different channel environment. Furthermore, based on the case study of V2V communication in the above section, we perform two sets of simulations focusing on the MAC layer behaviors and compare the performance of EDCA with distributed coordination function (DCF). Finally, two lane changing schemes using BSMs and higher priority safety messages are proposed and evaluated using the proposed simulation environment.

4.1 Evaluation of VANET Toolbox

In this section, the performance of VANET Toolbox is analyzed and evaluated in terms of events numbers and the execution time with increasing number of vehicles. The code

optimization using C code generation and profile features is also discussed.

4.1.1 Computational Costs in terms of Events

A detailed simulation of the whole vehicular network stack is time-consuming especially with a large amount of vehicles to simulate large-scale vehicular communication effect. In this section, we show the computational costs in term of number of events E in a discrete event-based simulation model.

Suppose we have n vehicles in a vehicular communication scenario, with each vehicle transmitting data at rate r in Hertz, and the simulation time is t in seconds. When a vehicle is willing to send a message, events are scheduled in order to generate the message in the APP layer and forwarded to the MAC layer, where the message is converted to a frame, experiences channel sensing, backoff, and finally sent to the PHY layer, all of which are conducted by different events. In the PHY layer, the frame is transformed into a waveform and sent into the wireless channel by events. After receiving a waveform, events are called in order to extract the information from the waveforms and send it way up to the APP layer and process it in the mobility models. We assume the number of events E per transmission is e. The number of events E per simulation can be calculated by Eq. (4.1).

$$E(n, r, t) = (nr) \cdot e \cdot (n - 1) \cdot t. \tag{4.1}$$

It is obvious to observe that the number of events E is linearly with the simulation time t and the data rate r, but nonlinear to the number of vehicles n. Figure 4.1 shows the number of simulated events for each layer of the vehicular network and for the overall communication set. We choose the car following model (CFM) as the scenario because for CFM, each vehicle broadcasts only BSMs at rate of 10 Hz and no other transmissions are involved. For a 600 second simulation, when 4 vehicles are involved, the total number of events is around $2 \cdot 10^6$. When vehicle number increases to 36, the total number of events is around $1.5 \cdot 10^8$, an increase by a factor of 75. The reason we present computational cost in terms of events is because we do not wish to be dependent on the choice of computing processing resources.

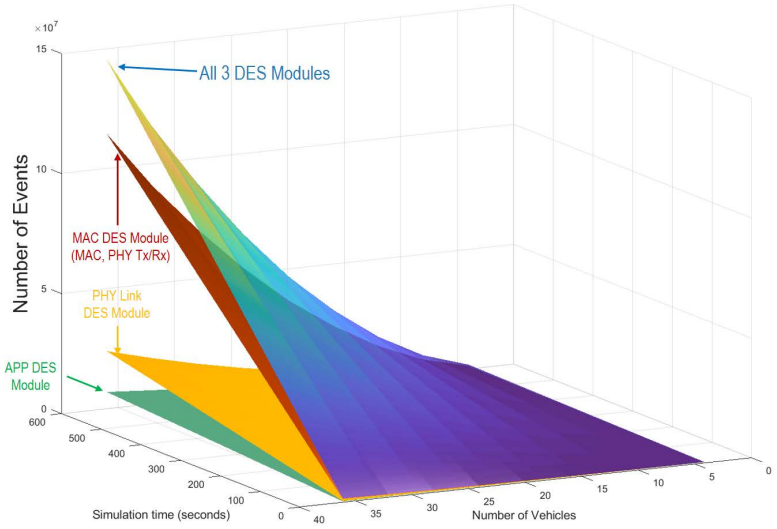


Figure 4.1: Computational cost in terms of events for each layers.

Among all the layers, the APP layer costs the least number of events since it is the top layer of the network stack and it mainly deals with message generation and reception. In our case, BSM is the only application data generated in the APP DES. If more applications are involved or the data generation rate is increased, the number of events will be increased accordingly. The PHY link DES involves slightly more events relative to the APP layer because whenever the APP DES generates one message and when this message enters the PHY link DES module in the format of a waveform entity, it triggers a series of events to deal with activities such as delay, buffer and waveform check. Thereby the number of events in the PHY link DES correlates to the number of messages generated in the APP DES module. The number of events increases dramatically in the MAC DES module. This is because the number of events in the MAC DES module is not only correlated to the number of APP layer messages but also affected by the channel status. Suppose if the wireless channel is congested, the channel sensing operation would be performed more frequently in order to monitor the channel status and seek a transmission opportunity. Thereby, the timer event related to the channel sensing operation is called frequently, which might cause a burst amount of events in the MAC DES module. In addition, the PHY Tx and Rx functions are

integrated with the MAC/PHY DES Module, the events caused by PHY Tx/Rx are shown in the MAC/PHY DES module surf instead of PHY Link DES module surf.

4.1.2 SimEvents Code Generation

MATLAB is a high-level interpreted type language and provides an interactive environment for numerical computation. The built-in math functions are able to reach a numerical solution more convenient than with spreadsheets or traditional programming languages, such as C/C++ or Java [98]. The tradeoff, however, is the sacrifice on the execution time compared with lower level compiled type languages such as C/C++. It may be not so obvious to notice how slow MATLAB language is when running simple functions or scripts. A vehicular network is usually in a large scale. The computational cost on number of events has been discussed above. It would be extremely slow when running vehicular network simulations with pure MATLAB/Simulink code.

Since 2017b, *SimEvents Toolbox* has been able to generate C/C++ code for a large portion of their classes including MATLAB DES. It is worth pointing out that *VANET Toolbox* is developed by SimEvents Toolbox and WLAN System Toolbox, and both toolboxes support code generation. Due to the limitations of code generation mechanism, WLAN System Toolbox is not code generated, only **major** components of *VANET Toolbox* developed by MATLAB DES of SimEvents are code generated. Saying 'major' means several peripheral functions, such as the local database and global database with persistent variables mentioned in Chapter 3, are not supported by code generation. All the functions are not code generated greatly slow down the execution speed. We will discuss more details in *Simulation Profile* subsection.

In order to test the performance of SimEvents code generation, two sets of experiments are conducted. The experiment scenario is a highway traffic with *simulation time* of 10 seconds. The number of vehicles increases from 5 to 35. One set is with SimEvents code generation and another is without SimEvents code generation. The results are shown in Figure 4.2. The computational testbed possesses the following characteristics: i7-6700k at 4.0GHz (CPU), 32G DDR4 2133MHz (memory),Microsoft Windows 10 (OS).

In the figure, the execution time increases as the number of vehicles increases. The

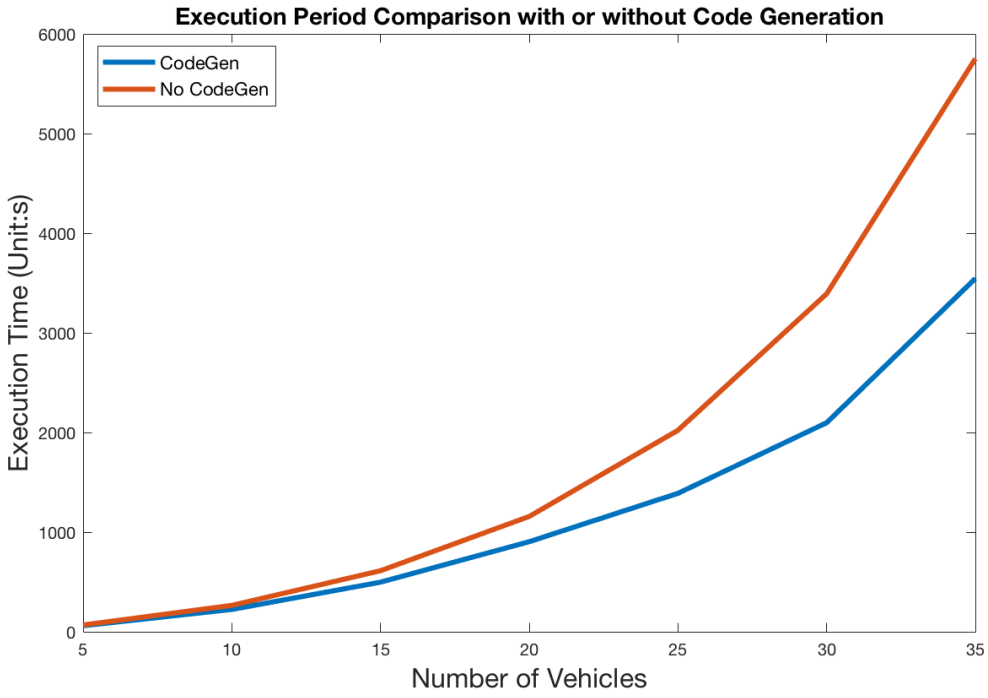


Figure 4.2: Comparison of execution time when running simulations with and without code generation. The simulation time is 10 seconds, the number of vehicles increases from 5 to 35.

red line indicates the simulations running in *interpreted execution* mode, *i.e.,* without code generation. The blue line is the execution time with code generation. It is obvious that after vehicle density is above 15, SimEvents code generation starts to show its advantage. For the simulation with 35 vehicles, *interpreted execution* costs 5751 seconds, while *code generation execution* costs 3545 seconds. The execution speed of 35 vehicles simulation increases 62.21% due to SimEvents code generation.

Simulation Profile

Profiler in MATLAB is able to track execution time for each functions. Knowing the execution time of the MATLAB functions help users locate the target code needed to be debug or optimized. We perform *profile* on a 30-second simulation with 30 vehicles. Partial results are shown in Figure 4.3.

Function Name	Calls	Total Time	Self Time
phy_waveform2psdu	310708	4336.688 s	991.357 s
fcn_carLocalDB	585786	1470.750 s	894.696 s
app_wsmp2msg	1811682	1162.013 s	101.107 s
num2str	59903153	1039.848 s	329.642 s
wlanNonHTDataRecover	309664	1007.626 s	116.578 s
int2str	59903123	710.200 s	710.200 s
fwdSubsref	15192330	569.265 s	569.265 s
phy_psdu2waveform	21430	511.383 s	10.432 s
AWGNChannel>AWGNChannel.stepImpl	310708	420.892 s	332.543 s
wlanSymbolTimingEstimate	310708	388.183 s	36.296 s
wlanBCCDecode	309664	351.671 s	16.162 s

Figure 4.3: Profile a 30-second simulation. The simulation includes 30 vehicles on the highway scenario with car-following mobility model. The highlighted four base functions cost most of the execution time.

The overall execution time costs 8535 seconds. Figure 4.3 shows major functions sorted from the longest total time to the lowest. It is worth point out that some functions are the child functions of other parent functions. For example, *num2str* and *int2str* are all child functions from *phy_psdu2waveform* or *app_wsmp2msg*. The top four parent level functions that consume most of time are listed below:

1. *phy_waveform2psdu*: This function is used to decode a received waveform (Data or ACK) to a message. It is implemented by *WLAN System Toolbox* on bit level. As it does not support code generation, the interpreted bit-level processing costs most of the execution time.

2. *fcn_calLocalDB*: This function is the local database for each vehicles. It contains large amount of calculations on the persistent variables. As described above, persistent variables do not support code generation. And other methods to maintain states, such as object properties, will prevent the overall code generation of MATLAB DES.

Therefore, there is nothing we can do on this file to accelerate the execution speed on current version.

3. *app_wsmp2msg*: This function convert an encoded WSMP to a regular message the applications can understand. The major component slows down the simulation is the *num2str*, which converts a numerical type data to a string type data. To convert a numerical WSMP to a string message, *num2str* is necessary but it does not support code generation. Therefore, there is nothing we can do on this file to accelerate the execution speed on current version.

4. *phy_psdu2waveform*: This is the reverse function of *phy_waveform2psdu*, which converts a MAC layer frame to waveform symbols. It is also implemented by *WLAN System Toolbox* on bit level. This function does not support code generation, thus the bit-level processing takes up a lot of time resources.

The profile above shows an opportunity to speed up the simulations. Both *phy_waveform2psdu* and *phy_psdu2waveform* consumes 57% execution time and they both supports code generation. However, due to the code generation limitations mentioned above, both functions need to make a lot of modifications to fulfill the requirements of code generation.

4.2 Evaluation of Vehicular Network

The vehicular inter-communication concept is motivated by improving road safety and efficiency. Safety related applications benefit directly from vehicle-to-vehicle (V2V) communication such as accident prevention applications, lane changing applications. Therefore, the vehicular network applications can be classified into safety-related applications and efficiency (Non-safety) applications. In this section, we only focus on V2V communication. All vehicles are autonomous, *i.e.,* their behaviors are fully based on V2V communication. The performance of V2V communication is evaluated by a series of experiments starting from the PHY layer and stacking to the full stack network with mobility models. That is the above layer activity always includes the below layers. For instance, the evaluation of the MAC layer activity includes both the MAC layer and the PHY layer, only the evaluation

focus is on the MAC layer logical activity. Similarly, the evaluation of the APP layer includes the APP layer, the MAC layer, and the PHY layer but the experiments are focusing on the APP layer activities.

4.2.1 Performance of the PHY Layer Activity

In this section, the performance of the MATLAB PHY layer is evaluated and compared with the NIST error rate model of NS-3, which is broadly adopted by iTETRIS and VSimRTI projects.

Precise PHY Layer Modeling

NS-3 is a packet-based, discrete-event network simulator equipped with several wireless models. When a waveform is received, NS-3 calculates the signal to noise ratio (SNR) and invokes its error rate model to decide the packet successful reception rate. Two error rate models are integrated with NS-3: the YANS [99] model and the NIST [49] model. The YANS model, which is based on an analytical bound was replaced with NIST error model in 2010. In this paper, we only focus on the currently used NIST error rate model.

In order to estimate the Packet Success Rate (PSR) for orthogonal frequency division multiplexing (OFDM) symbols, NIST calculates $\frac{E_b}{N_o}$ (SNR per bit E_b to the one-side noise spectral density N_o) based on SNR in dB using :

$$\frac{E_b}{N_o} = SNR - 10log10(k), \tag{4.2}$$

where $k = log_2(M)$, M is the modulation level, and k is the number of bits per symbol. However, the NIST error model has two limitations.

First, the NIST error model does not consider the oversampling situation and equates SNR to $\frac{E_s}{N_o}$, *i.e.,* ratio of symbol energy to noise power spectral density. The relationship between $\frac{E_s}{N_o}$ and SNR in dB for complex signals is defined by [100]:

$$\frac{E_s}{N_o} = SNR + 10log_{10}\frac{T_{sym}}{T_{samp}}, \tag{4.3}$$

where T_{sym} is the symbol period of the signal and T_{samp} is the sampling period of the signal. For a complex baseband signal, if it is oversampled by a factor of n, then $\frac{E_s}{N_0}$ does not equals to SNR but exceeds by $10log_{10}(n)$.

Second, the NIST error model does not account for the energy in nulls. Take IEEE 802.11p for instance, an OFDM signal consists of 64 subcarriers, among which 48 subcarriers are for data, 4 subcarriers for pilot information and 12 subcarriers are NULL [70]. Thus, the SNR for occupied subcarriers in dB, SNR_o is calculated using :

$$SNR_o = SNR - 10 * log_{10}\frac{N_{FFT}}{N_{data} + N_{Pilot}}, \tag{4.4}$$

where N_{FFT} is the number of FFT sampling points, *i.e.*, the total number of subcarriers for a OFDM signal. N_{data} is the quantity of subcarriers used for data and N_{pilot} is for pilot.

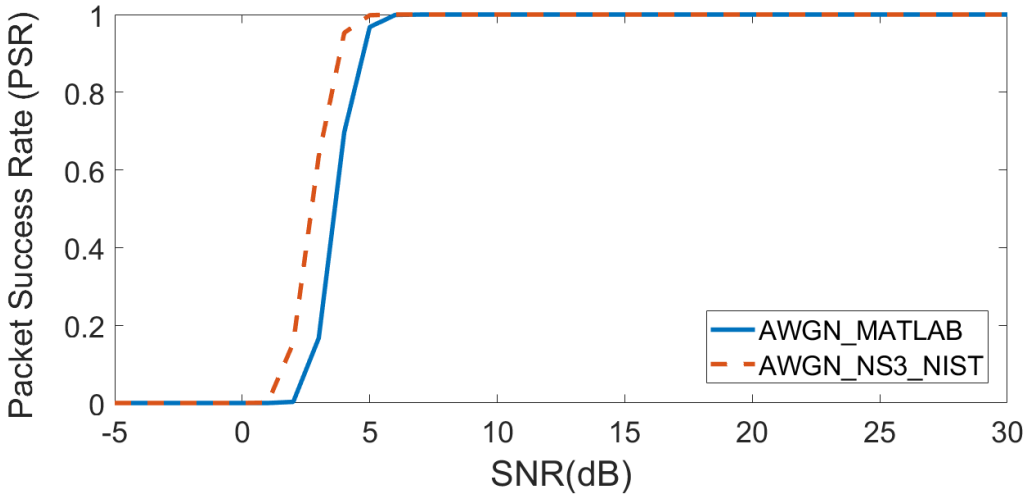


Figure 4.4: The packet success rate (PSR) comparison between MATLAB and NS-3 (NIST model). The simulation is conducted with BPSK modulation in AWGN channel.

Figure 4.4 shows the comparison of PSR on AWGN channels between NIST error model and proposed MATLAB error model. The PSR of the NIST error model is over optimistic while the proposed MATLAB error model is more realistic. As a packet-based network simulator, NIST model is the most NS-3 can implement. The oversampling situation as well as the energy in null subcarriers requires processing on bit level, thus NS-3 is difficult to implement these features. The AWGN channel shown in Figure 4.4 is a simple scenario

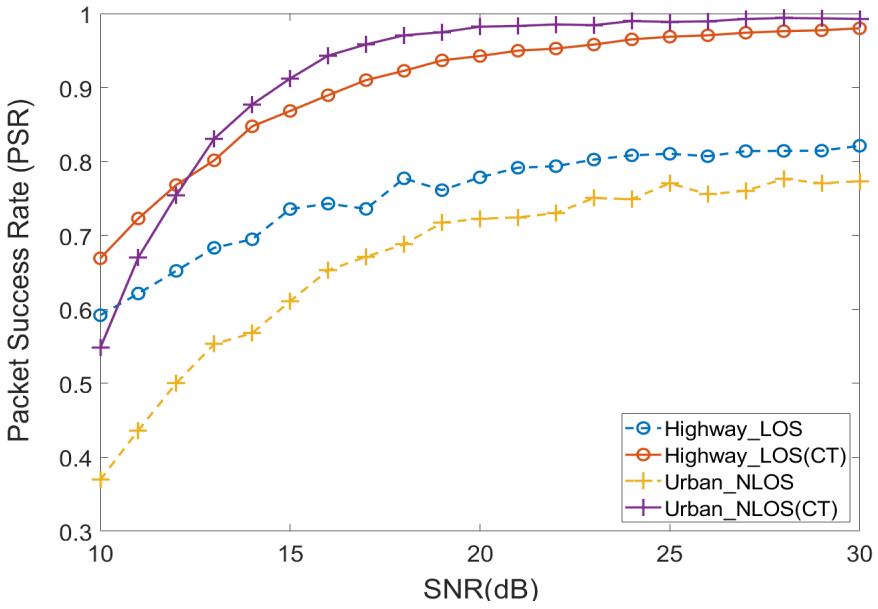


Figure 4.5: The performance of channel tracking (CT) for BPSK modulation in multi-path fading channel. The channel models involves highway line-of-sight (LOS) and urban non-line-of-sight (NLOS).

that can be compensated by incorporating an offset to NS-3 simulator. However, in a more complicated environment, such as Non-line-of-sight (NLOS) in Urban scenario, a constant offset is insufficient to cope with different channel models.

PHY Layer on Bit-level Processing

Another limitation of NS-3 is the oversimplified PHY layer. The packets are forwarded among objects of *Packet class* via methods. Based on the packet-error rate (PER) obtained from *NIST_Error_Model*, NS-3 randomly corrupts received packets in order to emulate the packet corruption process. However a realistic PHY layer of a communication system consists of more functions including frequency offset correction, channel estimation, modulation, and demodulation. The proposed simulator implements all the PHY layer features at the bit level.

In a vehicular network environment, the V2x channels have different characteristics compared with other stationary indoor channels [101]. First, V2x channels are affected by longer

multipath fading, which increases the possibility of intersymbol interference (ISI). Second, the transmission environment is highly dynamic, which causes significant Doppler effects resulting in more channel fading. When passing through the V2x channels, the waveforms are impacted more than just passing through an AWGN channel. The performance in terms of PSR will be degraded, thus channel tracking techniques are needed in order to enhance the performance.

In the proposed simulator, we integrate a time and frequency selective multipath Rayleigh fading channel as specified by [102] with an AWGN channel. The conventional WLAN channel estimation from L-LTF is used for the entire packet duration. In order to compensate the high Doppler spread of the V2x channel, channel tracking is enabled. With channel tracking, the channel estimation obtained from L-LTF is updated per symbol using decision directed channel tracking as presented in [101, 103]. We compare the performance in terms of PSR on BPSK across different scenarios including Highway LOS and Urban NLOS with channel tracking (CT) on and off. The results are presented in Figure 4.5. In this figure, the receiver with channel tracking (CT) enabled possesses a better PSR in V2x channels. Due to the restrictions of NS-3 on packet-level processing, implementing channel tracking to OFDM symbols is relatively difficult. It is worth mentioning that the focus of the paper is presenting the ability and accuracy of bit-level processing and using simple channel model to evaluate for straightforward comparison. More complicated channel models may be implemented in the future research.

4.2.2 Performance of the EDCA MAC Layer Activity

One significant feature of a vehicular network is adopting EDCA in the MAC layer. In this section, we first evaluate the EDCA scheme to assess the performance when data with different priorities coexist in the same channel. Then we compare the performance between EDCA and distributed coordination function (DCF), *i.e.,* carrier sensing media access (CSMA), where the latter is generally used in other IEEE 802.11 products. Research in [104] evaluates the performance of CSMA on highway scenario. In this section, we conduct simulations of both CSMA and EDCA on highway scenario. By analyzing the simulation results, we can evaluate the effectiveness of the proposed simulator.

We implemented three V2V models with 10 vehicles, 20 vehicles and 30 vehicles respectfully. For each vehicle, two traffic profiles are used. The first profile consists of BSM messages with AC2 priority and transmitted at a constant rate of 10Hz. The second profile consists of safety-critical messages generated using a Poisson process with $\lambda_{ec3} = 2$. Safety-critical message has the top priority, *i.e.,* AC3 priority [105].

All messages are in a fixed length of 100 bytes in both AC2 and AC3 priorities. The range of transmission is 300 meters. The density of vehicles, β, is defined as number of vehicles per meter:

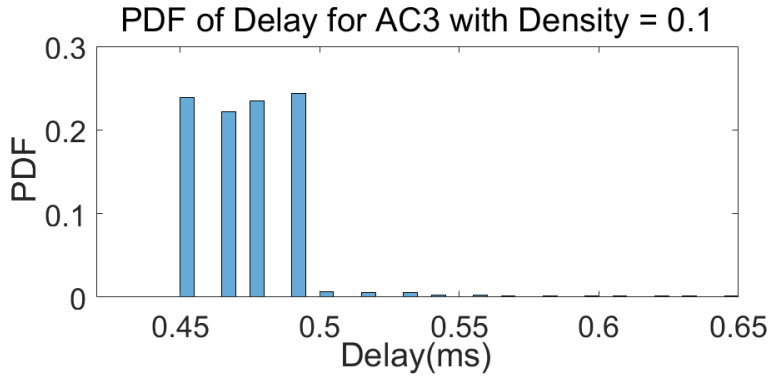
$$\beta = \frac{\{number\ of\ vehicles\}}{\{transmission\ distance\}}$$

Therefore, for vehicle numbers are 10, 20, 30, the corresponding β is 0.03, 0.07, and 0.1. Due to space limitation, only statistics on 10 and 30 vehicles are presented. The primary parameters used in the simulations are shown in Table 4.1.

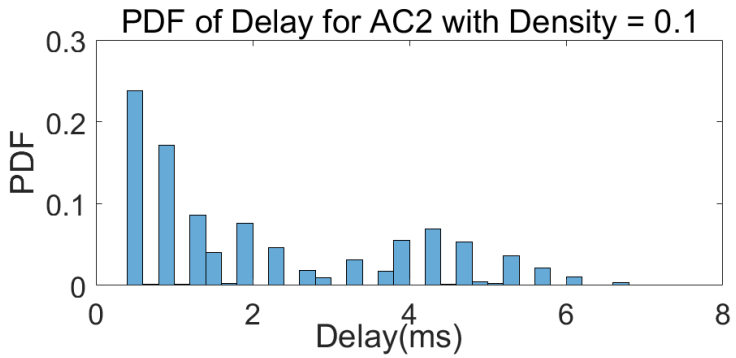
Table 4.1: Simulation Parameters for all simulations.

Parameter	Value
Transmission Range	300m
Packet Size	100 bytes
Density(β)	0.03 to 0.1 car/m
Data rate	3Mbps
Simulation time	100s

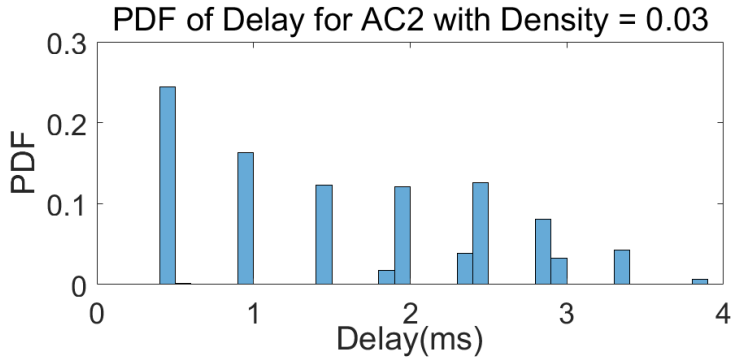
In this paper, we mainly focus on the distribution of the delay in the V2V network. The metric *delay* is defined as the period from the moment when the message is created to the moment that the receivers receive it correctly. The statistical results are shown in Figure 4.6 and Figure 4.7 and we compare the performance of our model with [78]. The figures show that the exponential distribution of the delay is as expected. The PDF of delay in Figure 4.6(a) and the CDF in Figure 4.7 show the delay distribution of AC3 when density is 0.1. We observe that the messages in AC3 with density of 0.1 manage to keep within 90% of the 0.6 ms of delay. This result shows us that the emergency messages could potentially



(a) Emergency msg- AC3 (density = 0.1)



(b) Routine msg- AC2 (density = 0.1)



(c) Routine msg - AC2 (density = 0.03)

Figure 4.6: PDF of the delay of messages from AC2 and AC3 with different traffic densities. The delay of AC3 is relatively concentrating around 0.47 ms. The delay of AC2 is more scattered compared with AC3 plot. As the traffic density increases, the delay increases accordingly.

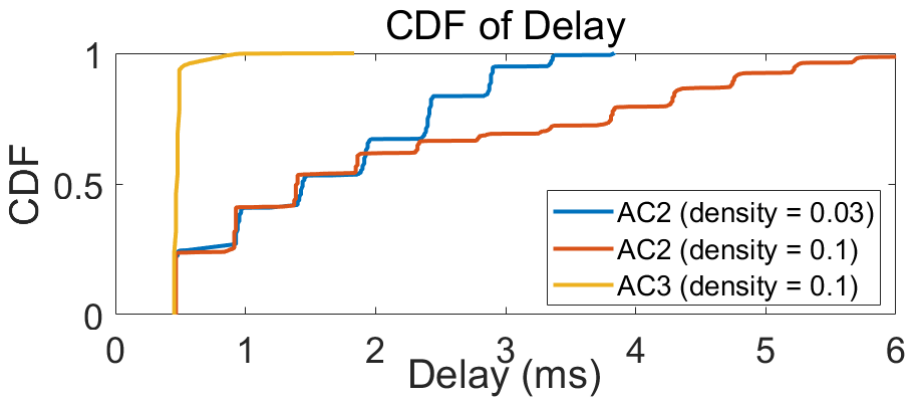


Figure 4.7: CDF of the delay of messages from AC2 and AC3 with different traffic densities. With the same traffic density (0.1), AC3 enjoys a more stable delay than AC2. With the increase of the traffic density from 0.03 (blue curve) to 0.1 (red curve), the spread of the delay becomes wider.

coexist in the same channel of BSMs if the QoS is respected by all the stations.

Conversely, messages in AC2 experience a longer delay as shown in Figure 4.6(b) and 4.6(c). Figure 4.7 shows that our simulation is within 80% of the delay when the density is 0.03, is around 2.5 ms and 4 ms when the density increased to 0.1. As the BSMs in AC2 are transmitted at a constant rate of 10Hz, they are more easily affected when the density increases.

Figure 4.8 shows another metric of network performance, packet delivery rate (PDR), the ratio number of packets delivered with no CRC error and the total number of transmitted packets. In the figure, packets from AC3 maintains a relatively stable level on PDR. As the density increases to 0.1, the PDR of AC3 only decreases by approximately 2%. This proves that with EDCA, the performance of the packets with the AC3 priority may be able to operate at a satisfactory level.

However, the PDR of AC2 drops dramatically as the vehicle density increases spectacularly. It drops from 80% with $\beta = 0.01$ to 10% with $\beta = 0.1$. The drop of PDR comes from the flood broadcast of BSMs. As the BSMs are broadcasted in 10Hz for each vehicle, when the number of vehicle in the coverage area increases, the broadcast may overwhelm the carrier sensing ability of the vehicle, thus the probability of collision will also increase accordingly, which causes the drop of the performance.

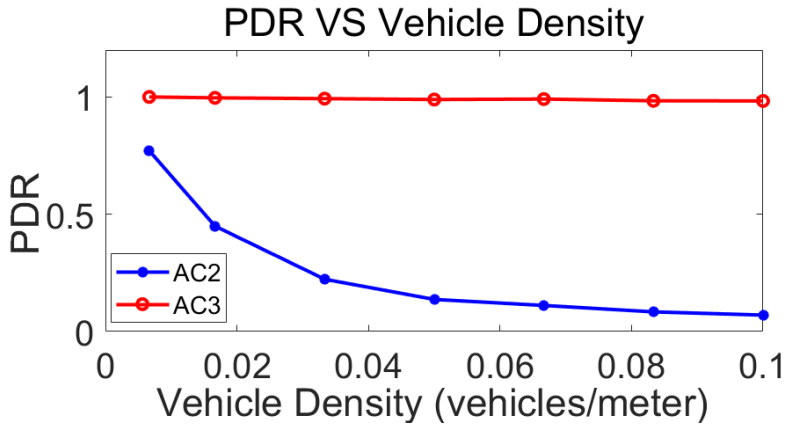


Figure 4.8: PDR of AC2 and AC3 with different traffic densities. With the increase of the traffic density, the PDR of AC3 maintains at the relatively high level, while the PDR of AC2 drops tremendously.

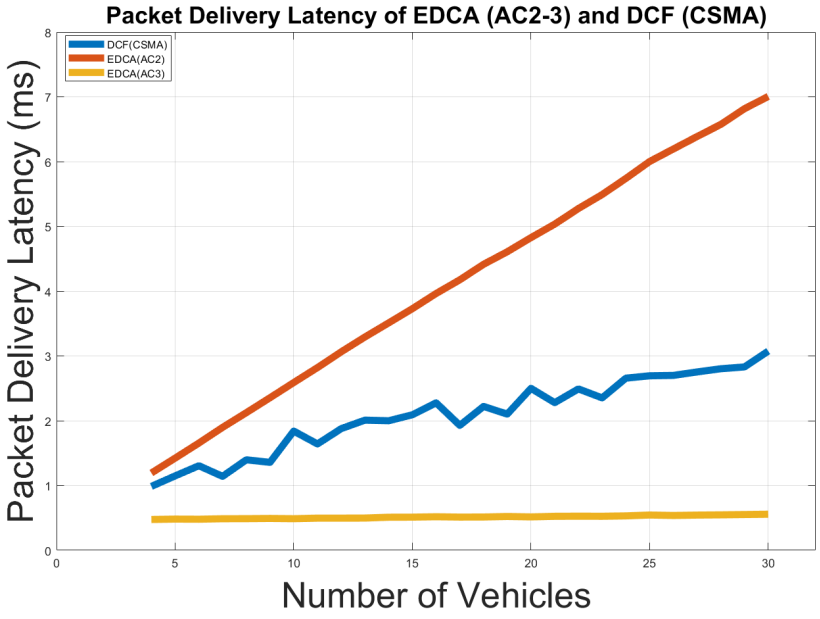
Table 4.2: Parameters of CSMA and EDCA in the simulations.

Parameters	CSMA (802.11a)	EDCA (802.11p)
IFS	2	3 (AC2) & 2(AC3)
SIFS (μs)	16	32
slot time (μs)	9	13
[CWmin,CWmax]	[15,1023]	[7,15] (AC2), [3,7] (AC3)

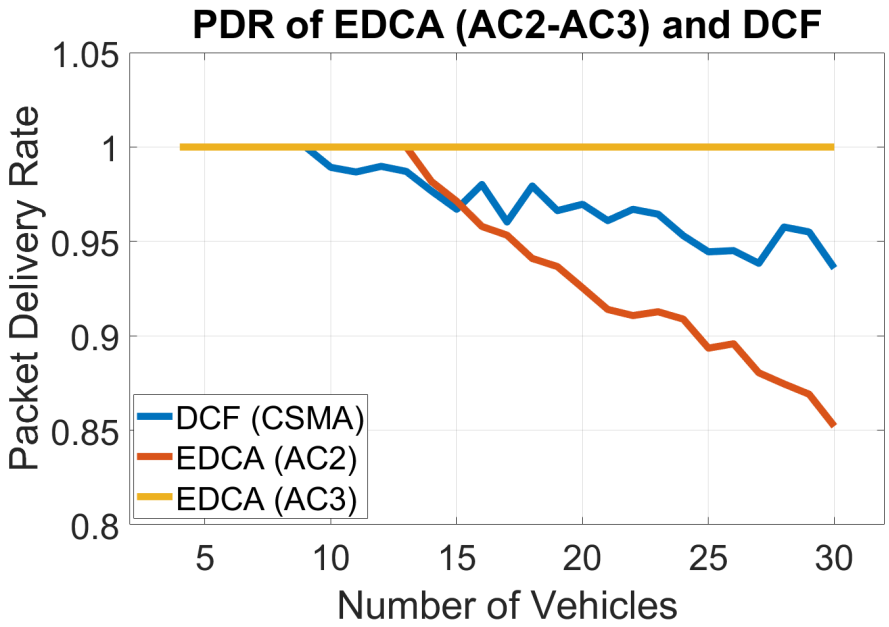
Performance Comparison between EDCA and DCF (CSMA)

One significant feature of a vehicular network is adopting EDCA in the MAC layer. In this section, we will compare the performance between EDCA and distributed coordination function (DCF), *i.e.,* carrier sensing media access (CSMA), where the latter is generally used in other IEEE 802.11 products. By analyzing the simulation results, we can evaluate the effectiveness of the proposed simulator.

The simulation scenario is a unidirectional highway consisting of two lanes. The car following model (CFM) is chosen to be the mobility model. The vehicles are broadcasting BSMs (AC2) at 10 Hz and AC3 messages using a Poisson distribution modeled by $\lambda = 2$. We performed two sets of simulations for EDCA and CSMA with the key MAC parameters



(a) Packet Delay of EDCA (AC2), EDCA (AC3) and DCF (CSMA).



(b) Packet Delivery Rate (PDR) of EDCA (AC2), EDCA (AC3) and DCF (CSMA).

Figure 4.9: Performance Evaluation of DCF (CSMA) and EDCA (AC2-3)

listed in Table 4.2. As the PHY layer of IEEE 802.11p is derived from IEEE 802.11a, we choose the IEEE 802.11a version of CSMA to minimize the difference with other layers.

Figure 4.9 shows the simulation results of the Packet Deliver Latency (PDL) and Packet Deliver Rate (PDR). As all the messages in the simulation are broadcast, there are no retransmissions involved, with the latency mainly coming from the channel access deference process, *i.e.,* IFS + backoff. In Table 4.2, the Inter-Frame-Space (IFS) of CSMA, *i.e.,* DIFS, equals to the highest priority IFS (AC3), *i.e.,* AIFSN(AC3), and slightly smaller than AIFSN (AC2). The Shortest-IFS (SIFS) and slot time of IEEE 802.11p is greater than IEEE 802.11a in order to cope with the mobility characteristics of the vehicular network. Only a minimum Contention Window (CW) is used for broadcasting purpose.

In Figure 4.9(a), the latency of CSMA is smaller than EDCA (AC2) but greater than EDCA (AC3). This is due to the fact that data possessing different priorities are queued into different ACs, while in CSMA all data are buffered in the same queue. Whenever an internal collision happens, AC2 always gives way to AC3. This is why AC3 shows a steady and better performance in the figure. According to SAE J2735 [74], the maximum latency for safety messages is 10 ms. When the latency of AC2 is below the threshold, EDCA is shown to be the better option than CSMA.

Fig. 4.9(b) compares the Packet Delivery Rate (PDR) of CSMA with AC2 and AC3 transmissions of EDCA. When the number of vehicles increases, the PDR of AC3 maintains at nearly 100 percent. This proves the AC2 and AC3 can coexist in the same channel, and AC2 traffic affects little on AC3 traffic. On the other hand, the PDR of AC2 starts to decrease when number of vehicles approaches to 15 due to the packet collisions. When less than 15 vehicles, EDCA (AC2) still performs better than CSMA. However, when more than 15 vehicles are present, CSMA acts better than EDCA (AC2). This is due to the coexistence of AC2 and AC3, which increases the packet collision rate. However, for 30 vehicles, the PDR of EDCA (AC2) is till above 85 percent, which performs well enough on broadcasting BSMs.

4.2.3 Performance of the APP Layer Activity

The basic function for the APP layer is to generate messages for different applications. As our proposed simulator is an *integrated* type simulator, *i.e.,* the vehicular mobility traffic flow models are integrated directly with the APP layer module in order to support the real-time simulation, the behavior of the APP layer is closely related to the mobility traffic flow models. The traffic flow models have been mentioned in Figure 3.8 in Chapter 3. We repost the flow models here, as in Figure 4.10. Vehicle i is the target vehicle who will perform car following or lane changing behaviors. At time t, the x position and velocity of vehicle i are represented as $x_i(t)$ and $v_i(t)$ respectfully. Vehicle $i-1$ and $i+1$ are the vehicles immediately behind and in front of vehicle i with x positions $x_{i-1}(t)$ and $x_{i+1}(t)$, and with speed $v_{i-1}(t)$ and $v_{i+1}(t)$. $\Delta d_i(t)$ indicates the distance from vehicle i to vehicle $i+1$ at time t. On the adjacent lane, the immediately back and front vehicles are denoted as vehicle $j-1$ and $j+1$. Similarly, their positions and speeds at time t are shown as $x_{j-1}(t)$ and $x_{j+1}(t)$, and $v_{j-1}(t)$ and $v_{j+1}(t)$. The major notations are summarized in Table 4.3.

Table 4.3: Notations used in the Highway mobility model.

notation	Description
Δt	time step in [seconds]
Δd	distance to the immediately front vehicle in [meters]
acc	acceleration in [m/s^2]
dec	deceleration in [m/s^2]
L	length of vehicle in [meters]
v_x	x velocity in [m/s]
v_y	y velocity in [m/s]
t_r	vehicle reaction period in [seconds]

Car Following Models (CFMs)

The CFMs control the individual vehicle's driving dynamics to maintain a safe distance to the vehicle immediately ahead. The objective of CFMs is to model vehicular traffic flows

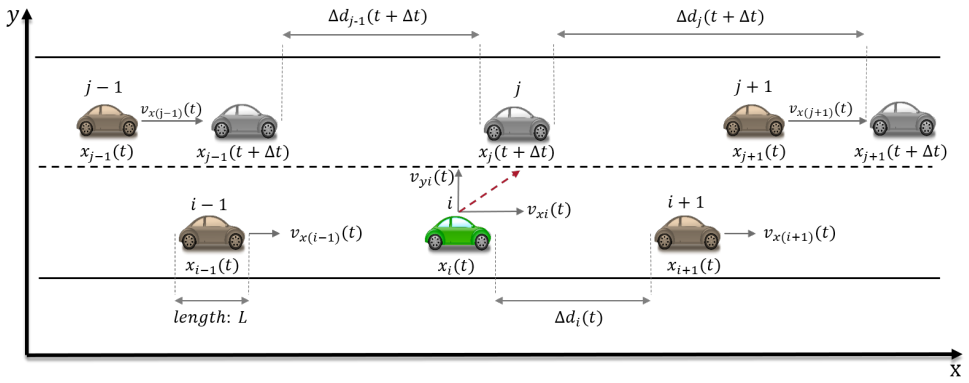


Figure 4.10: Notations for Highway Mobility Model based on V2V Communications.

without car collisions under the help of vehicular communication. The safe distance of vehicle i is generally calculated by Eq. (4.5):

$$d_{i,safe} = L_i + t_{react} * v_{xi}(t) + d_{brake}. \tag{4.5}$$

$$d_{brake} = \frac{v_{xi}(t+\Delta t)^2 - v_{xi}(t)^2}{2 \cdot \mu_s \cdot dec}\bigg|_{\mu_s=1, v_{xi}(t+\Delta t)=0}. \tag{4.6}$$

$$d_{brake} = \frac{-v_{xi}(t)^2}{2 \cdot dec}. \tag{4.7}$$

$$dec_k = -\mu_k * g = -0.8 * 9.8 = -7.84 m/s^2. \tag{4.8}$$

where L_i is the length of vehicle i. t_{react} is the reaction time either of the driver or from the autonomous vehicular dynamics. d_{brake} is the braking distance and is calculated in Eq. (4.6). $v_{xi}(t)$ is the instantaneous speed when brake is performed, and $v_{xi}(t+\Delta t)$ is the velocity when braking action is finished, for a complete stop, $v_{xi}(t+\Delta t) = 0$. Therefore, we have the full stop brake distance as shown in Eq. (4.7).

The deceleration, dec, is determined by the current vehicle speed, road surface friction coefficient μ, as well as the friction type, *i.e.,* static friction and kinetic friction. If a vehicle is driving on a dry concrete road surface, according to [106], the static friction coefficient is $\mu_s = 1$ and the kinetic friction coefficient is $\mu_k = 0.8$. When the vehicle brakes free and

slides, the kinetic friction deceleration dec_k only depends on μ_k and the acceleration due to gravity $g = 9.80m/s^2$, shown in Eq. (4.8).

For static friction, NHTSA in [107] shows a mapping between speed and braking distance, based on which we set the maximum static friction deceleration to $dec_s = -6.50m/s^2$. For $dec \in [dec_s, 0]$, we define this type of braking as *regular brake*. For $dec < dec_s$, the brake action is called as *emergency (EMG) brake* and $dec = dec_k$.

In CFMs, the vehicle keeps monitoring the distance to the vehicle immediately ahead of it based on the received BSMs and adjusts speed adaptively in order to maintain a safe distance. When the front vehicle is braking, the vehicle behind it will be aware of it through the BSM information and start to brake. More sophisticated CFMs have been proposed including Intelligent Driver Model (IDM) [108] and Wiedemann Model [109]. These models can be implemented and used in *VANET Toolbox* if necessary.

Lane Changing Models (LCMs)

LCMs are based on multi-lane traffic and needs to be studied. A general LCM should include three parts: the *trigger* for a lane changing, the *feasibility* of a lane changing and the *scheme* used during lane changing process.

Consider the situations and notations shown in Figure 3.8, a basic lane changing scenario involve four vehicles. The *trigger* for vehicle i to change lane is the distance to vehicle $i+1$ is shorter than the safe distance, i.e., when $\Delta d_i(t) < d_{i,safe}$.

The lane changing *feasibility* of vehicle i is determined by the distance to vehicle $j-1$, i.e., $\Delta d_{j-1}(t+\Delta t)$, and the distance to vehicle $j+1$, i.e., $\Delta d_{j+1}(t+\Delta t)$ during the lane changing process. Both distances should not violate the safe distance in order to avoid potential car collisions on the adjacent lane when changing lane. The speed boundary for vehicle i is calculated as follows:

$$|\frac{0^2 - v_{x,i}(t)^2}{2 \cdot dec_s}| < |\frac{0^2 - v_{x,j+1}(t)^2}{2 \cdot dec_s}| + d_{i,j+1}(t). \tag{4.9}$$

$$|\frac{0^2 - v_{x,i}(t)^2}{2 \cdot dec_s}| + d_{i,j-1}(t) > |\frac{0^2 - v_{x,j-1}(t)^2}{2 \cdot dec_s}|. \tag{4.10}$$

$$max(v_{x,i}(t)) = \sqrt{v_{x,j+1}(t)^2 + 2 \cdot |dec_s| \cdot d_{i,j+1}(t)}. \tag{4.11}$$

$$min(v_{x,i}(t)) = \sqrt{v_{x,j-1}(t)^2 - 2 \cdot |dec_s| \cdot d_{i,j-1}(t)}. \tag{4.12}$$

The lane changing prediction algorithm assumes the vehicle shift to the adjacent lane at a y-speed of $v_{y,i}(t)$ while adjusting the x-speed $x_i(t)$ during the process. Eq. (4.9) calculates the upper speed boundary of vehicle i. During the lane changing process, vehicle $j+1$ is possible to perform braking and the shortest braking distance is determined by dec_s. $d_{i,j+1}$ is the distance on x axis before lane changing. The lane changing algorithm should predict if $d_{i,j+1}$ is a safe distance for a lane change, i.e., $d_{i,j+1}(t)$ should be greater than the difference of the brake trails from both vehicles. Similarly, the lower speed boundary is calculated by Eq. (4.10). The maximum and minimum speeds for vehicle i during lane changing are shown in Eq. (4.11) and Eq. (4.12) respectively.

If the lane changing *feasibility* is not fulfilled, vehicle i has to reduce its speed $v_{x,i}(t)$, i.e., brake, to respect the safety constraint. If the *feasibility* is allowed, vehicle i will start to change lane. The *trajectory* model includes lane changing period, target lane chosen etc. More advanced lane changing models such as Nagel-Schreckenberg model [110], Krauss (1998) Model [111] can be implemented and selected by *VANET Toolbox* if necessary.

In a vehicular network, the vehicles share their traffic information via safety related messages. The formats of these messages are defined in SAE J2735 [74], which specifies 15 types of safety messages including the Basic Safety Message (BSM), as well as the Signal Phase Time (SPT) and MAP message [112]. According to SAE standards, BSMs are sent out periodically at the rate of 10Hz and it is mandatory in DSRC that all connected vehicles need to broadcast to the surrounding vehicles. BSMs contain vehicle state information as shown in Table 4.4

In order to coordinate lane changing among the involved vehicles, we created a new type of message, namely the *Lane Changing Message (LCM)*. LCMs include *Lane Changing Request* and *Lane Changing Reply*. LCMs are assigned AC3 priority while BSMs are assigned AC2 priority. In [113], we evaluated the performance of the proposed *VANET Toolbox* and

Table 4.4: The Information of Basic Safety Messages (BSMs).

Items	Explanation
Latitude	1/10 micro degree precision range ± 90°
Longitude	1/10 micro degree precision range ± 180°
BrakeSystemStatus	Structure of brake system status per wheel and ABS
Acceleration	Acceleration in 3 orthogonal directions
SteeringWheel	Angle of steering wheel in 1.5° step

showed that AC2 and AC3 data can coexist on the same channel in heavy traffic conditions.

The proposed lane changing schemes are based on V2V communications. In order to simplify the problem, we assume that a single lane change involves four vehicles, as shown in Figure 4.10. Car 1 plans to change lanes and is aware of the existence of car 3 and car 4 from the BSMs, then coordination is needed among cars 1, 3, 4. Coordination is implemented via the LCMs exchanged ahead of the lane changing actions. LCMs have AC3 priority and require the use of reliable data transmission (RDT), *i.e.,* Data-ACK mode. RDT is implemented across the MAC layer.

In an RDT, an ACK is expected after sending data. If the ACK is not received within a predefined period, the data will be retransmitted until the ACK is received or the maximum retransmission limit is reached. If the ACK is not received until after the maximum number of retransmissions, the data is discarded. Due to this reason, a timer for the lane changing requests on the APP layer is necessary. Since a vehicle cannot always wait for a lane changing reply if there is a transmission failure on lane changing request, a timer on the APP layer can let the vehicle cancel the lane changing plan and brake in time. According to WAVE 1609.4 [17], the timeout for one pair of Data-Ack RDT is 2.605ms considering retransmissions and channel access delays. In our proposed work, we set the timeout to 5ms because both lane changing schemes include two sets of RDTs. Suppose a vehicle is driving at a speed of 120km/h, it only moves 16.67cm within 5ms, which does not have significant impact on braking distance. In other words, even if a vehicle failed to receive lane changing reply within the period, it still has enough time to brake.

The Packet Drop Rate (PDR) can potentially increase as the vehicle density increases. The lost of packets, such as BSMs, may prevent vehicles from obtaining the latest traffic information from other vehicles. The lane changing schemes we proposed have considered this situation and provided reliability to the lane changing behaviors. The reliability is guaranteed using feedback from other relevant vehicles. We name the proposed lane changing schemes into *performance lane-changing* and *conservative lane-changing*, as shown in Figure 4.11 based on different message exchange patterns.

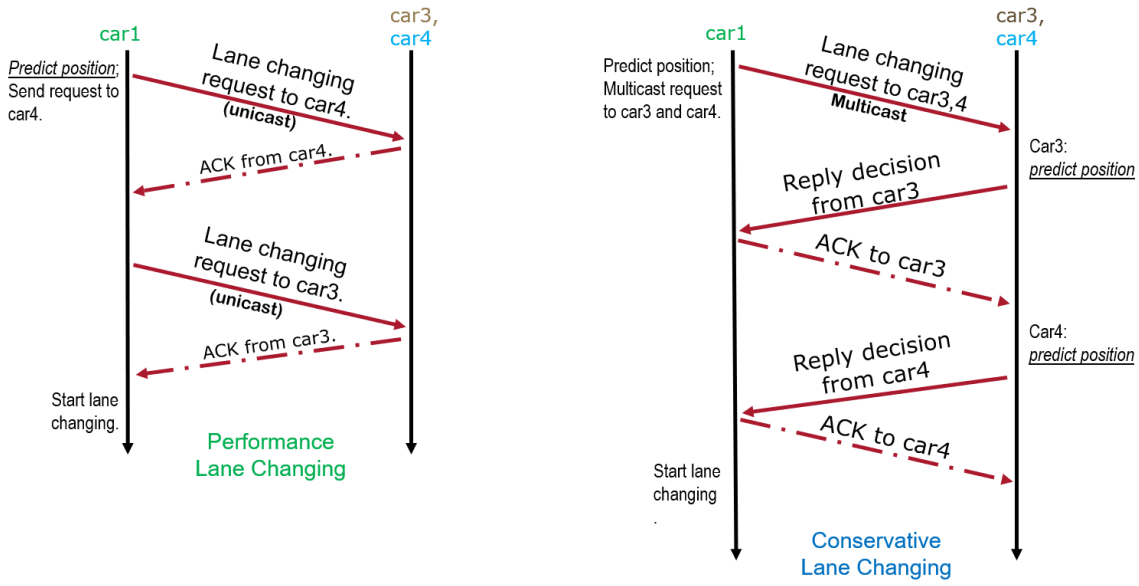


Figure 4.11: Lane changing messages exchange process of PerLC and ConLC. PerLC dedicates to shorten the message exchange period and ConLC is to guarantee the safety of lane changing.

Performance Lane Changing Scheme (PerLC)

For PerLC (refer to Figure 4.11, left), car 1 predicts the traffic condition based on the information from the BSMs and evaluate the feasibility of the lane changeing. Once the condition permits, car 1 sends its RDT lane changing requests to car 3 and car 4, respectively. Only when car 1 receives ACKs from both car 3 and car 4 can car 1 start to change lanes. The received ACKs indicate that car 3 and car 4 are aware of the lane changing behavior of car 1. After receiving the lane changing requests from car 1, then car

3 and car 4 will try their best to avoid potential car accidents by performing the necessary actions including deceleration. The PerLC scheme is designated to shorten the message exchange period among the vehicles involved during the lane changing process and provide a relative high reliability for lane changing actions. However, we notice the lane changing decision is only made by car 1 based on the BSMs information received at the beginning of the stage. Car 3 and car 4 have no chance to deny the lane changing request from car 1. In other words, car 1 does not possess the real permissions from car 3 and car 4 in order to change lanes regardless if their conditions are allowed for a safe lane change (the ACKs are sent back automatically).

Conservative Lane Changing Scheme (ConLC)

In order to compensate the weak feedback of PerLC, ConLC is proposed, as shown in Figure 4.11-right. Car 1 predicts the traffic condition before lane change, which is similar to PerLC scheme. Once the prediction indicates a safe lane changing opportunity, car 1 starts to multicast a lane changing request with multicast address including car 3 and car 4. Once receiving the multicast request, the surrounding vehicles check if its carID is included in the multicast address. If included, it means that this vehicle is required to coordinate with car 1 on lane changing. Otherwise, the multicast request is disregarded. In the scenario shown in Figure 4.10, the multicast request enables car 3 and car 4 to evaluate whether the lane changing behavior is feasible or not, as well as sending back the answer in the *lane changing reply* message. Only when car 1 receives both positive replies from car 3 and car 4, can it start to change lane. Otherwise, car 1 should continue waiting for permissions until timeout, then starts to brake. The advantage of ConLC is the vehicles doubles the insurance to guarantee the safety during the lane changing process and car 3 and 4 can then piggyback extra information on the lane changing replies. This information can help car 1 make a more reliable lane change. The tradeoff of this process is that the message exchange period is also longer than PerLC.

The reason why PerLC has no multicast is because it is not suitable for RDT,and that the ACKs from the receivers possess a high probability with respect to collision. As mentioned in the previous section, the feedback from other vehicles is necessary to guarantee the

reliability of lane changing schemes. Therefore, unicast with ACK is chosen for PerLC. In the next section, the performance results of these two lane changing schemes are evaluated.

Scenario Model

In this section, we evaluate the performance of both lane changing schemes and compare the results with a vehicle-following trajectory. The experiment consists of a set of computer simulations. Each simulation is 30 seconds long and is performed across a 1600-meter section of unidirectional highway. All messages, including BSMs and LCMs in the simulation, are at a fixed size of 100 bytes. The number of vehicles is increased from 4 to 15 and each case runs with 10 repetitions. Each repetition has the same initial position but random initial speed in a normal distribution with mean of 70 km/h and standard deviation of 5 km/h. The parameters used in the simulations are shown in Table 4.5.

Table 4.5: Simulation Parameters for all simulations.

Parameters	Values
Packet Size	100 bytes
Vehicle Density	4 - 15
Data rate	3 Mbps
Simulation time	30s

The purpose of the proposed lane changing schemes is to obtain a better flow of vehicles across the road. One of the metrics to describe the traffic flow is the average speed across all the vehicles. If the average overall speed is higher, the traffic flow is more efficient.

Non-Lane Changing (NonLC) scenario is selected as a baseline in the experiments, *i.e.*,, all vehicles are only allowed to brake if necessary. Figure 4.12 shows the average overall speeds for PerLC, ConLC and NonLC with error bars at 96% confidence interval (CI). In Figure 4.12 , the average speed decreases as the vehicle density increases from 4 to 15, which is reasonable. When vehicle density is smaller than 10, the performance of PerLC and ConLC is overlapped. This is due to the communications being sufficient and possesses minimal impact on both PerLC and ConLC schemes. When the density is larger than 11,

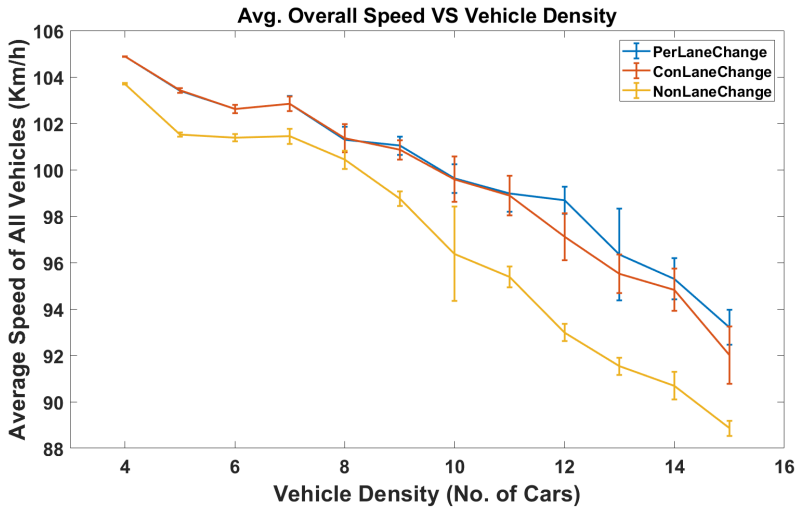


Figure 4.12: Average speed of all the cars on the road over increase of density of car. PerLC has a better performance on average traffic speed than ConLC at medium traffic density. Both lane changing schemes are better than NonLC.

PerLC starts to demonstrate its advantage as the wireless channel starts to become crowded, ConLC experiences more latency during lane changing exchange period. But both PerLC and ConLC exhibit much better performances related to NonLC, which provides that lane changing schemes do improve the overall traffic efficiency.

In order to show how the vehicle density affects the performance of V2V communication, we gather the message exchange delay for both PerLC and ConLC schemes over different vehicle densities. Figure 4.13 shows message exchange latency with the error bars at 96% CI. As the vehicle density increases, the V2V communication channel becomes congested. A busy channel may causes longer channel access deference and/or more retransmissions. This is why the message exchange latency for both lane changing schemes increases.

When vehicle density is larger than 9, the error bar also increases, which shows a significant variation in the message exchange latency. Different sets of initial speeds lead to varieties of traffic patterns, and they have significant impact on lane changing frequencies. For example, multiple vehicles might be changing lanes within a short period simultaneously such that it causes a very busy channel during this period, which might result in time the lane changing message exchange may experience severe delay. However, PerLC maintains a

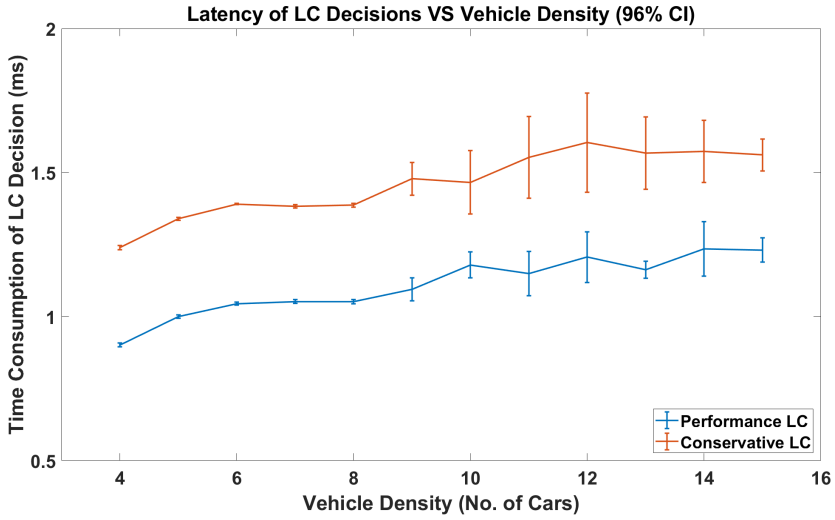


Figure 4.13: Average message exchange latency between PerLC and ConLC over Vehicle Densities. Both latencies increase as the channel becomes busier, PerLC is more time efficient than ConLC.

much lower latency than ConLC on all vehicle densities.

4.2.4 Comparisons between Proposed Simulator and NS-3

NS-3 is a well developed discrete-event network simulator that supports full stack standards on varieties of networks. The NS-3 modules are robust and can perform a faster speed due to its C++ source code. However, NS-3 might have some limitations. First, its PHY layer is packet-based, the minimum data element is on packet level instead of bit level. Thus the bit related operations such as channel tracking, channel estimation and frequency offset correction cannot be applied. Besides, NS-3 is lack of supports on real radio hardware as it is unable to convert information into bits or symbols. Second, NS-3 was originally designed as a pure network simulation environment. In order to simulate vehicular traffic, NS-3 either uses predefined route information or interacts with mobility simulators asynchronously with the help of interfaces. The randomness of vehicular traffic scenario might not be able to simulate in real-time.

Our proposed simulation environment makes up the limitations of NS-3. First, it provides a more accurate PHY layer on bit level. The simulator is able to simulate the channel

impairments such as noise, path loss or shadow fading to the bits. Besides, the bits are converted into symbols, which are exactly the same format in the real wireless communications. Thus it is reasonable to infer that the simulated wireless channel can be replaced by the real software defined radios (SDR) such as USRP. The real radio transmission may be evaluated in the future research. Furthermore, as the mobility models are integrated with the APP layer, the reciprocal interactions between the traffic application and the network communication are well supported. This feature makes the proposed simulator to simulate vehicular driving operations and network communications synchronously in real time.

Table 4.6: Comparisons of features and limitations between proposed simulator and NS-3

	VANET Toolbox
Features	1. More accurate bit-level PHY layer simulation. 2. Expandable to SDR hardwares and other toolboxes. 3. Support traffic simulation in real time. 4. Support multi-OS: Windows, Linux, OSX.
Limitations	1. Parameters are fixed during the simulation. 2. Relative slow simulation speed. 3. MATLAB/Simulink and toolboxes are not free. 4. Still under development, bugs may exist.
	NS-3
Features	1. Full stack simulation on varieties of network types. 2. Well developed comprehensive simulation modules. 3. Relative fast simulation speed. 4. Free software, open source
Limitations	1. Packet-based PHY layer, less accurate, 2. limited hardware radio expandability. 3. Limited support on real-time traffic simulation. 4. Primary running in Linux, less supports in other OS.

On the other hand, the proposed simulator has some limitations. First, MATLAB is an interpreted-type programming language and aiming for precisely modeling. Compared with

other compiled-type programming languages such as C++, used in NS-3, fast execution speed or less computational cost is not MATLAB's advantage. Especially, the PHY layer of *VANET Toolbox* is designed on bit level processing, the execution speed is even slower than NS-3. Furthermore, *VANET Toolbox* does not support parallel computing, *i.e.,* the model created by *VANET Toolbox* or its parent *SimEvents* cannot be run over multiple cores. Even though *SimEvents* supports C/C++ code generation since R2017b, it still has several limitations. For instance, C/C++ code generation does not support hash map, persistent variables, or changing the values of properties of an object inside another object. Those functions have to be declared as extrinsic functions and cannot enjoy the benefit of C/C++ code generation. Thus the relatively slow execution speed is the major limitation of the proposed simulator. Table 4.6 summarized the major features and limitations for both simulators.

4.3 Chapter Summary

In this chapter, we first tested the eligibility of the proposed vehicular network simulator. The computational costs in terms of events is evaluated. The reason we present computational cost in terms of events is because we do not wish to be dependent on the choice of computing processing resources. Partial code of the simulator can be converted into C code in order to increase the execution speed. The code generation is able to save up to 62.21% of execution time than pure MATLAB code. MATLAB is an interpreted type programming languages, the execution speed is not MATLAB's advantage compared with other compiled type programming languages such as C++, which is used to develop NS-3. However, the goal of developing models with MATLAB is to create an accurate PHY layer on bit level. Thus the bit-level PHY layer of the proposed simulator is compared with packet-level simulator, NS-3. The simulation results prove that the bit-level PHY is able to generate a more precise channel model by controlling each symbol of the waveform. In addition, channel tracking towards the L-LTF of a OFDM symbol can only be implemented by bit-level MATLAB code. Then the performance of EDCA scheme in the MAC layer is compared with the traditional CSMA scheme based on single channel V2V communication.

The simulation results show that the EDCA scheme can make data with different priorities coexist in the same channel. And the QoS of data with higher priority with respect to lower latency and higher packet delivery ratio can be guaranteed. This feature is useful for safety-related services compared with CSMA scheme, where all messages are granted with the same priority. Finally, the lane changing model in the APP layer is designed with two coordinated message exchange patterns, *i.e.,* conservative lane changing (ConLC) and performance lane changing (PerLC). With the vehicular network communication, the simulation results indicate that coordinated lane changing operation can improve the overall traffic efficiency.

Chapter 5

Vehicular Network Simulation with Multichannel Operations

In this chapter, we describe how we extend the single channel simulation environment to support multichannel operations. Then the performance of safety-related services in both single channel and multichannel channel scenarios is evaluated in terms of packet delivery latency and packet delivery radio. Then we assessed the latency and throughput of non-safety services with different number of vehicles as well as data streams. Furthermore, we proposed a new SCH reservation mechanism, which allocate SCHs according to the mixed SCHs/ACs information conveyed in the enhanced WSAs within at most 2-hop distance.

5.1 Background of Multichannel Vehicular Network

Vehicular network based on standards [19, 80, 114] provides wireless access for vehicles to share messages. The messages are brought by varieties of services, which can be divided into safety-related services, *e.g.,* emergency brake, basic safety messages (BSMs) [115], and non-safety services including traffic efficiency services and infotainment services [116]. The safety-related services requires reliability with low latency, while the non-safety services prefer high throughput. IEEE 1609.4 [80] defines a prioritized Enhanced Distributed Channel Access (EDCA) that provide different level Quality of Service (QoS) to different services.

The EDCA module grants four Access Categories (ACs) within a node with different priorities according to the critical level. Our previous research efforts [113, 117] prove that EDCA is effective for multiple safety-related services with different priorities coexist in the same channel.

However, with more and more non-safety applications involved, single channel EDCA is insufficient to guarantee the reliability of safety-related services in terms of low latency and high PDR. Thus IEEE 1609 [80, 114] standards specify the multichannel option to enhance the QoS. In multichannel mode, the channel time is sliced into continuous 100 ms Synchronization Intervals (SIs). Each SI consists of a Control Channel interval (CCHI) and a Service Channel interval (SCHI) with equal length of 50 ms. The safety-related messages are transmitted via CCH during the CCHI while non-safety messages are transmitted via at least one SCH during the SCHI. In this way, the safety-related services are separated with non-safety services to maintain the reliability, while the non-safety services are shared by n SCHs for high throughput purpose. A vehicle with single radio is mandatory to alternate between the CCH and one of the SCHs. Unlike single channel scenario, where services have immediate channel access after backoff process, multichannel services have to wait for their corresponding channel intervals. This may potentially result in low channel utilization due to channel switching operation activity. Suppose the transmission of a safety service is not finished at the end of CCHI, it is paused for the adjacent SCHI (50 ms) no matter if there are transmissions from non-safety services, then the transmission is resumed in the next CCHI. Thus the channel utilization is less than 50% for the SI. Researchers from [118–121] evaluated the performance of multichannel. However, they either did not apply EDCA to the multichannel transmission, or only applies partial of EDCA, *i.e.,* not all ACs are activated during the transmission. Thus the results might be less convincing to evaluate the performance.

Another challenge is the IEEE 1609.4 standard [80] does not specify a criteria for SCH selection. Thereby multiple service providers may accidentally choose the same SCH frequency at the same time due to the hidden terminal problem [122]. Several research efforts have been underway to the SCH reservations. Similarly, they either do not consider EDCA [123, 124] or only use two out of four ACs [125, 126] with one for safety-related ser-

vices and the other for non-safety services. In this section, the experimental simulations are conducted with data streams of four AC priorities such that the impact to the safety-related services from the non-safety messages is observed and whether multichannel operation has the ability to enhance the performance of safety-related services is evaluated.

5.1.1 Multichannel PHY Layer

The US Federal Communications Commission (FCC) assigned 75 MHz of spectrum at 5.9 GHz to be used for vehicular communication networks [127]. In order to increase the tolerance for multi-path propagation effects in vehicular environments, the spectrum is divided into seven 10 MHz channels, *i.e.,* CH172 - 184, as shown in Figure 5.1. Vehicular communications are based on Orthogonal Frequency Division Multiplexing (OFDM), which divides the 10 MHz channel into 52 orthogonal sub-carriers. Compared with the at least 20 MHz Wi-Fi channel bandwidth, the 10 MHz channel reduces the effect of Doppler spread. Amongst the seven channels, CH178 is restricted to safety-relayed communications such as Basic Safety Messages (BSMs) and control message disseminations, such as Wave Service Advertisement (WSAs). The remaining frequencies are available for both safety and non-safety services. For instance, CH172 and CH184 are reserved for future usage of critical safety of life services and high power public safety communications.

In multichannel mode, the clock signal for all vehicles are synchronized by GPS. If a vehicle is not equipped with a GPS, the clock signal can be synchronized by the WAVE timing advertisement (WTA) message received from the other vehicles during CCHI. A WTA message provides the time difference from last transmitted UTS in microseconds. If no GPS signal or WTA message has been received, the vehicle will continue to stay in the CCH until its time can be synchronized by either GPS or received WTA message. The synchronized channel time is sliced into Synchronization Intervals (SIs), with each SI being 100 ms in duration in order to cope with the 10 Hz BSMs [120]. A SI is a repeating time interval comprised of a 50 ms Control Channel interval (CCHI) followed by a 50 ms Service Channel interval (SCHI). The first 4 ms of the CCHI or SCHI is a guard interval (GI), which is reserved for nodes switching among the channels. IEEE 1609.4 standard [17] supports the options of either having single radio multichannel operation or multiple radios multichannel

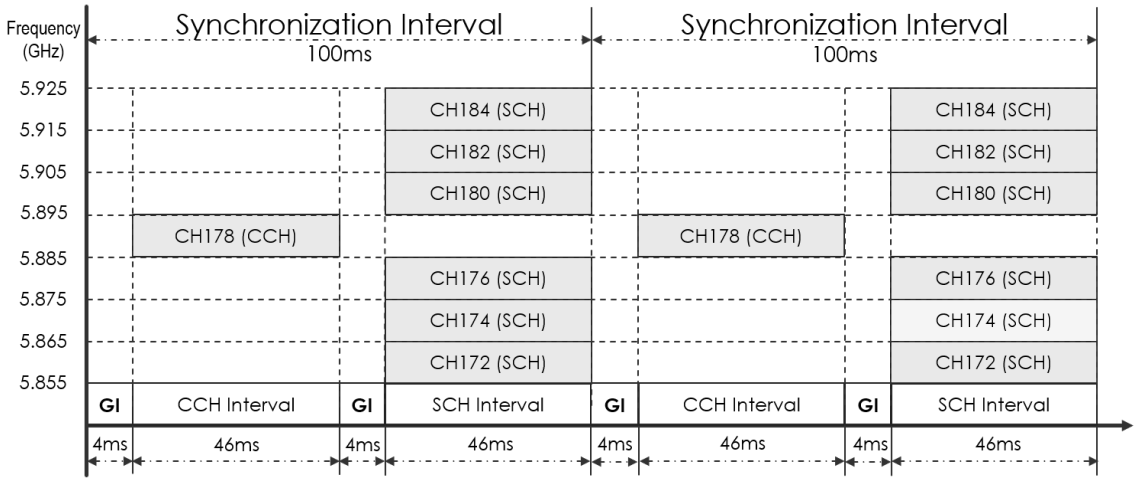


Figure 5.1: Spectrum allocation of vehicular network PHY layer. The 75 MHz is divided into 7 channels with 10 MHz bandwidth. The channel period is sliced into 100 ms synchronization intervals consisting 50 ms CCHI and 50 ms SCHI.

operation. In this paper, we only focus on the single radio multichannel scenario. Periodic switching between the CCH and SCH is required for the single radio multichannel scenario and operates on only one radio channel at a time [119]. All vehicular nodes are required to tune into the CCH (CH178) during the CCHI for the transmissions of safety-related messages or control messages. During the SCHI, the vehicles have the option to tune into one of the SCHs if they want to initiate or join a non-safety service.

5.1.2 Multichannel MAC Layer

The MAC layer of a vehicular network enables the Enhanced Distributed Channel Access (EDCA) mechanism [128], which is derived from the carrier sense multiple access (CSMA) with collision avoidance (CA) scheme. EDCA provides distributed channel access by granting data with one of four access categories (ACs). The ACs each possesses different sets of access parameters, including arbitration inter-frame space (AIFS[AC]) and contention window (CW[AC]) size values. When the data is at the head of an AC queue, it must wait for a period equal to AIFS[AC], which is calculated by Eq. (5.1), where the Short Inter-Frame Space (SIFS) is a fixed amount of time required to process a received frame and to send a response frame. After the AIFS period has elapsed, if the channel is sensed as idle,

the data will backoff by any amount equal to BF_n time slots, which is calculated by Eq. (5.2), where *randi* indicates a random integer between 0 and $CWmin[AC]$. If the channel is sensed busy, then the value of $CWmin[AC]$ is doubled but no larger than $CWmax[AC]$ as shown in Eq. (5.3). Then, a new BF_n is recalculated using Eq. (5.2) with the updated $CWmin[AC]$ value.

$$AIFS[AC] = SIFS + AIFSN[AC] \times slottime. \tag{5.1}$$

$$BF_n = randi(0, CWmin[AC]). \tag{5.2}$$

$$CWmin[AC] = min(2 \times CWmin[AC], CWmax[AC]). \tag{5.3}$$

It is possible the data from the different AC queues within the same node contend for the channel access simultaneously, which produces the situation called internal contention. In this case, the data with the higher priority will be sent to the PHY layer, while the data with the lower priority will need to repeat the backoff according to the updated values calculated from Eq. (5.1) - (5.3).

Table 5.1: The EDCA parameters for CCH and SCH MAC Modules.

	ACI	**CWmin**	**CWmax**	**AIFSN**
EDCA (CCH)	AC0	aCWmin	aCWmax	9
	AC1	(aCWmin+1)/2-1	aCWmin	6
	AC2	(aCWmin+1)/4-1	(aCWmin+1)/2-1	3
	AC3	(aCWmin+1)/4-1	(aCWmin+1)/2-1	2
EDCA (SCH)	AC0	aCWmin	aCWmax	7
	AC1	aCWmin	aCWmax	3
	AC2	(aCWmin+1)/2-1	aCWmin	2
	AC3	(aCWmin+1)/4-1	(aCWmin+1)/2-1	2

In the multichannel MAC layer, the EDCA is adopted to work in the multichannel environment by applying the multiple EDCA functions to one MAC module, with one function used for CCH and at least one function used for SCHs. Data to be sent to different PHY channels enter into the AC queues via different EDCA functions. Table 5.1 shows the default EDCA parameters for CCH and SCH [17, 119], where AC3 denotes the highest priority and AC0 denotes the lowest priority.

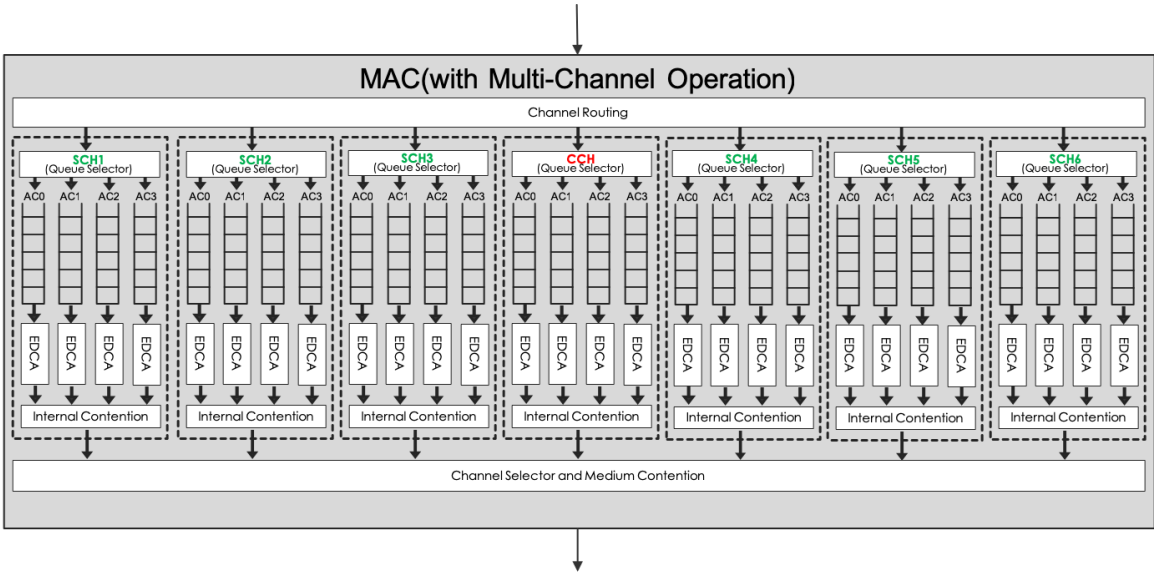


Figure 5.2: Multichannel EDCA MAC layer modules. Seven EDCA modules are created corresponding to one CCH and six SCHs. The design of multiple SCH modules allows a node to join multiple non-safety services over multiple SCHIs simultaneously.

Figure 5.2 shows the multichannel EDCA structure of the MAC layer. The payload from the APP layer enters the MAC layer and is forwarded to the corresponding EDCA AC queue set by the *Channel Routing* module based on the channel information from the payload. The EDCA AC sets are alternatively switching between the 'paused' and 'active' status values. Only one EDCA AC set is active at any time during a channel interval. When an EDCA AC set is active, the data performs backoff based on the parameters in Table 5.1. Meanwhile, the backoff processes for the other EDCA AC sets are paused. Once the backoff is completed, the *Channel Selector* module selects the corresponding radio frequency and transmits the data.

The reason why one CCH EDCA is chosen along with six SCH EDCA modules instead of having one CCH EDCA with one SCH EDCA module as stated in other research [119] is that we need to consider the situation where a vehicular node might become involved with more than one non-safety service possessing the same AC priority across different SCHs over multiple SCH intervals simultaneously. Suppose a vehicle is supporting service1 with AC2 in SCH1 and service2 with AC2 in SCH2. During the first SCHI, service1 is transmitted but some of the data cannot be transmitted at the end of that SCHI. The data is buffered in the SCH1 EDCA AC queues. Then, service2 occupies the next SCHI and the backoff activities are operated in the SCH2 AC2 queue. In the third SCHI, the data of service1 resumes backoff and continues transmission. Consequently, using multiple EDCA sets is necessary in order to separate data with the same AC priority but with different SCHs. Furthermore, this design greatly decreases the implementation complexity.

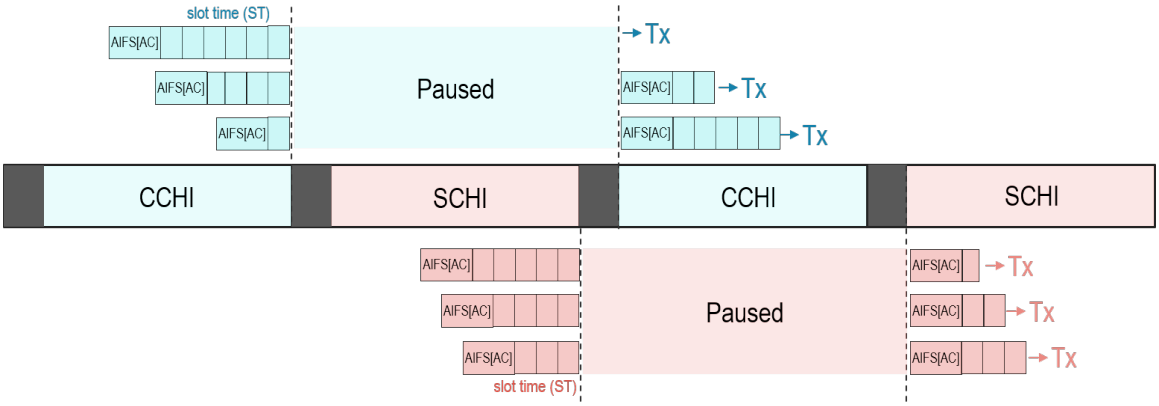


Figure 5.3: Impact to backoff process due to channel switching operation. If channel switching operation happens in the middle of process, the unfinished backoff processes are paused and resumed in the next synchronization interval.

When channel switching occurs in the middle of a transmission, *i.e.,* the GI commences prior to the completion of a transmission, this transmission has to be canceled. IEEE 1609.4 [17] suggests the vehicular network developer must create algorithms that can effectively schedule transmissions in order to avoid the aforementioned problem. The vehicular network system may either send the unfinished data back to its original AC queue until the next cycle or purge the expired data if they reach the time to live (TTL) period, such as BSMs [129]. If the GI occurs during the process of receiving, the reception is canceled by the receiver

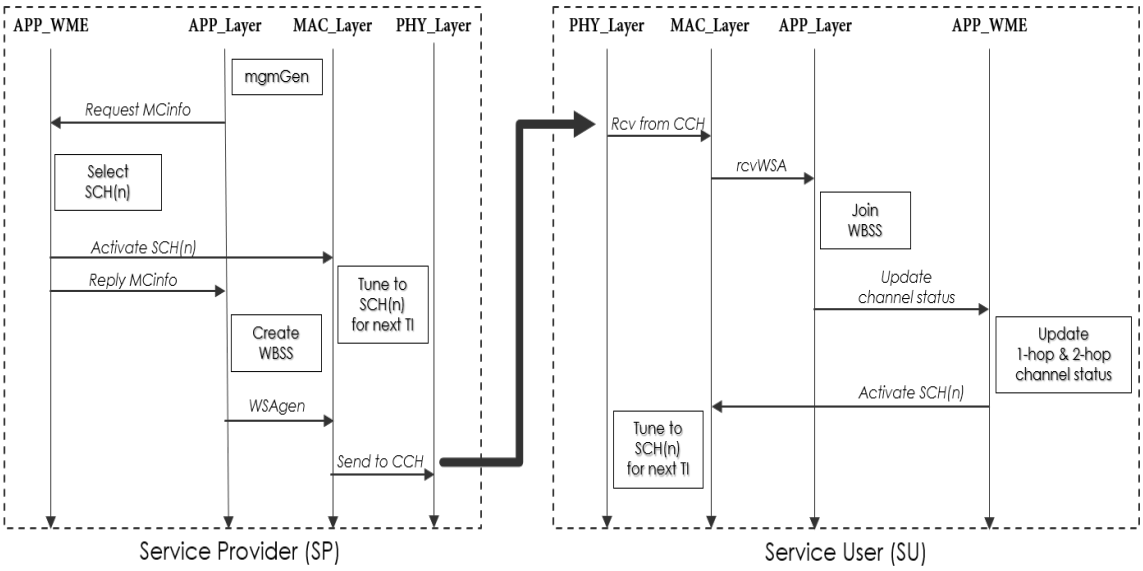


Figure 5.4: The establishment process of WBSSs with WSA coordination. Multichannel information is stored in the WME module and used for generating WSAs. The coordination process must be finished during CCHI before WBSSs are established in SCHI.

immediately [130].

If the channel switching occurs during the backoff process, as shown in Figure 5.3, the backoff is paused at the start of the GI. The time duration of the GI along with the following channel interval and the next GI, which totals 54 ms are treated as a busy channel. When the next corresponding channel interval becomes available, if the backoff process finishes exactly at the same time as the start time of GI but the transmission has not started yet, thus the data will be transmitted immediately at the beginning of available channel interval without any channel sensing. On the other hand, isf the backoff process is incomplete and paused due to channel switching, then it will resume with a channel sensing period of AIFS[AC] at the beginning of the next corresponding channel interval followed by the remaining backoff timeslots. This random backoff mechanism prevents nodes from sending simultaneously.

5.1.3 Multichannel Coordination via WSA

Non-safety services are shared within a WAVE Basic Service Set (WBSS) and it is established in an ad-hoc manner amongst vehicles without any authentication and association processes. The vehicles initiating the services are called service providers (SPs). The vehicles who are interested in these services are called service users (SUs). A SP usually creates a WBSS by announcing the service information such as service name, type, and serviceID via WAVE Service Advertisements (WSAs). After receiving a WSA, the SU tunes its radio frequency to the advised SCH frequency in the next SCHI in order to join the WBSS and access the service. Figure 5.4 shows the message exchange process involving a SP and SU when forming a WBSS. Since IEEE 1609.4 [17] does not specify how to assign service channel access for multiple requests from higher layers, we designed our own SCH reservation mechanism to select SCHs as well as update the SCH information in the WME. Additional details regarding our proposed SCH reservation scheme are provided in Section 5.3.4.

The service starts from the APP layer of a SP, which first creates a management message (*mgmGen*) and sends it to the WAVE management entity (*APP_WME*) module in order to query multichannel information (*MCinfo*). Each APP_WME module maintains a SCH vector, which stores the MCinfo including the SCH number and the AC priority. The SCH vector is updated either by the SPs when generating the WSAs in order to initiate the WBSSs or by the SUs once it is receiving the WSAs. In the case of Figure 5.4, the APP_WME module selects a SCH from the SCH vector and replies to the APP layer in a subsequent management message. Meanwhile, the MCinfo is sent to the multichannel MAC module informing it to activate the corresponding SCH in the next SCHI. The APP layer creates a WSA message with the MCinfo obtained from the APP_WME. The WSA is passed to the CCH_MAC submodule and is finally broadcasted to the wireless channel. If the SP is currently in a CCHI, the WSA is transmitted after the backoff process. If the SP is currently in a SCHI, the WSA will be sent in the next CCHI.

When receiving a WSA, the vehicular node extracts the payload of the WSA and sends it upward to the APP layer. If a vehicular node is interested in the service and wants to join the WBSS, it becomes a SU. The SU creates a management message to its APP_WME module in

order to update the SCH vector. With our proposed SCH reservation mechanism, the SCH vector can be updated with the SCH/AC information for a maximum distance of 2 hops. Similar to the APP_WME module of a SP, a management message with the SCH information is sent to the multichannel MAC module. Based on the SCH information, the MAC layer will activate the corresponding SCH_MAC module in the next SCHI. Consequently, a WBSS is established without any authentication and association process.

5.2 Implementation of Multichannel Operation

Figure 5.5 shows the full stack of a multichannel vehicular network designed using DES modules. The APP layer module is integrated with several mobility models such as the car-following model and the lane-changing model, which control the vehicle movements on the road. Additionally, the APP layer module is responsible for generating multiple types of messages, including safety-related messages, non-safety messages, and control messages. In this work, the APP layer messages are called *payload entities*.

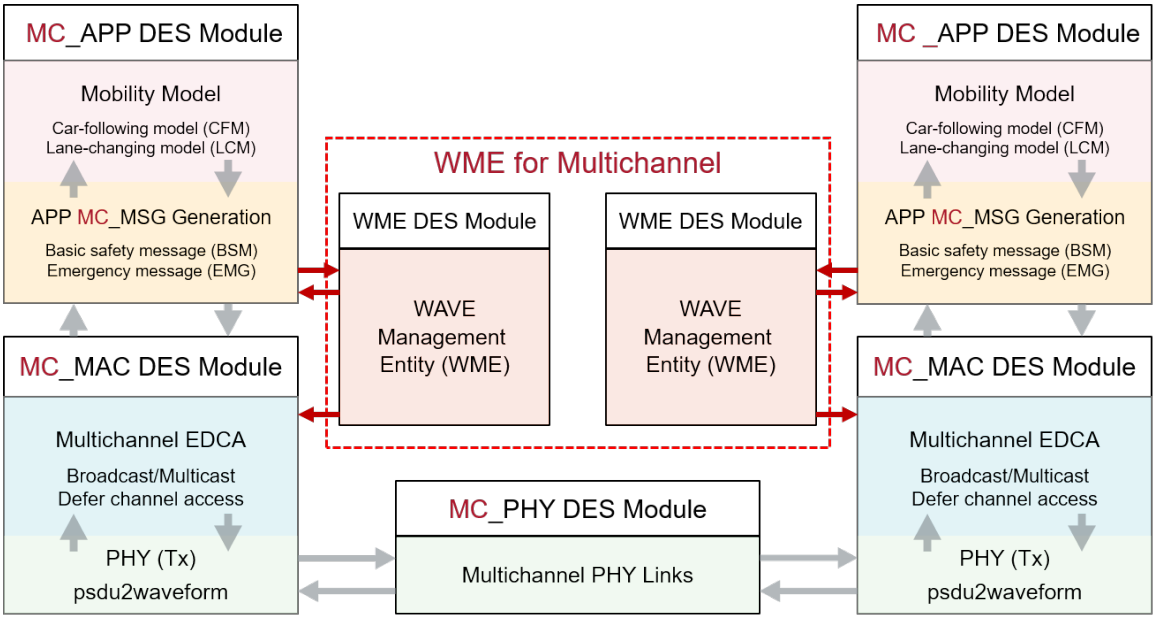


Figure 5.5: The stack of vehicular network in multichannel mode. WME module acts as the bridge between APP layer and MAC layer. APP layer obtains SCH information from WME to create WSA. MAC/PHY layer are tuned by WME module.

In the figure, the payload entities of Car 1 are forwarded to the MAC layer, where they are converted into *frame entities*. The frame entity who wins the internal contention is converted into a *waveform entity* and forwarded to the wireless channel. When the PHY layer module of another car receives the waveform entity from the wireless channel, it performs waveform integrity check. The corrupted waveforms will not pass the cyclic redundancy check (CRC) and are discarded. The payload information is extracted from the error-free waveforms and is sent to the APP layer of Car 2. The traffic information in the received message is applied to the mobility model and may affect the traffic actions of the vehicles.

In multichannel mode, all three layers are involved to support multichannel operations. An extra WME DES module is added to configure the multichannel switching and coordination. The details about the DES framework for a single channel vehicular network are provided in [113, 117]. For this paper, we only focus on the design of multichannel related features in DES. The library of VANET Toolbox is now updated to version 3.0 with newly designed multichannel blocks as shown in Figure 5.6. The functions of the newly added modules in VANET Library v3.0 are briefly described as follows:

1. APP_MC DES Module: Coordinates with WME module to obtain MC information and generates payload with MC information.

2. MAC_MC DES Module: Consists of multiple EDCA modules corresponds multiple channels and coordinates with WME module to activate/deactivated EDCA modules.

3. PHY_MC DES Module: Support multiple separate physical channels with different frequencies.

4. WME_MC DES Module: Maintains MC status and provides MC information to APP_MC and MAC_MC modules.

5. Vehicle_MC DES Module: Establishes WBSSs or joins WBSSs.

Fig. 5.7 shows the framework of a multichannel node designed in DES. In this figure, a vehicular node DES module consists of a multichannel APP DES module, multichannel

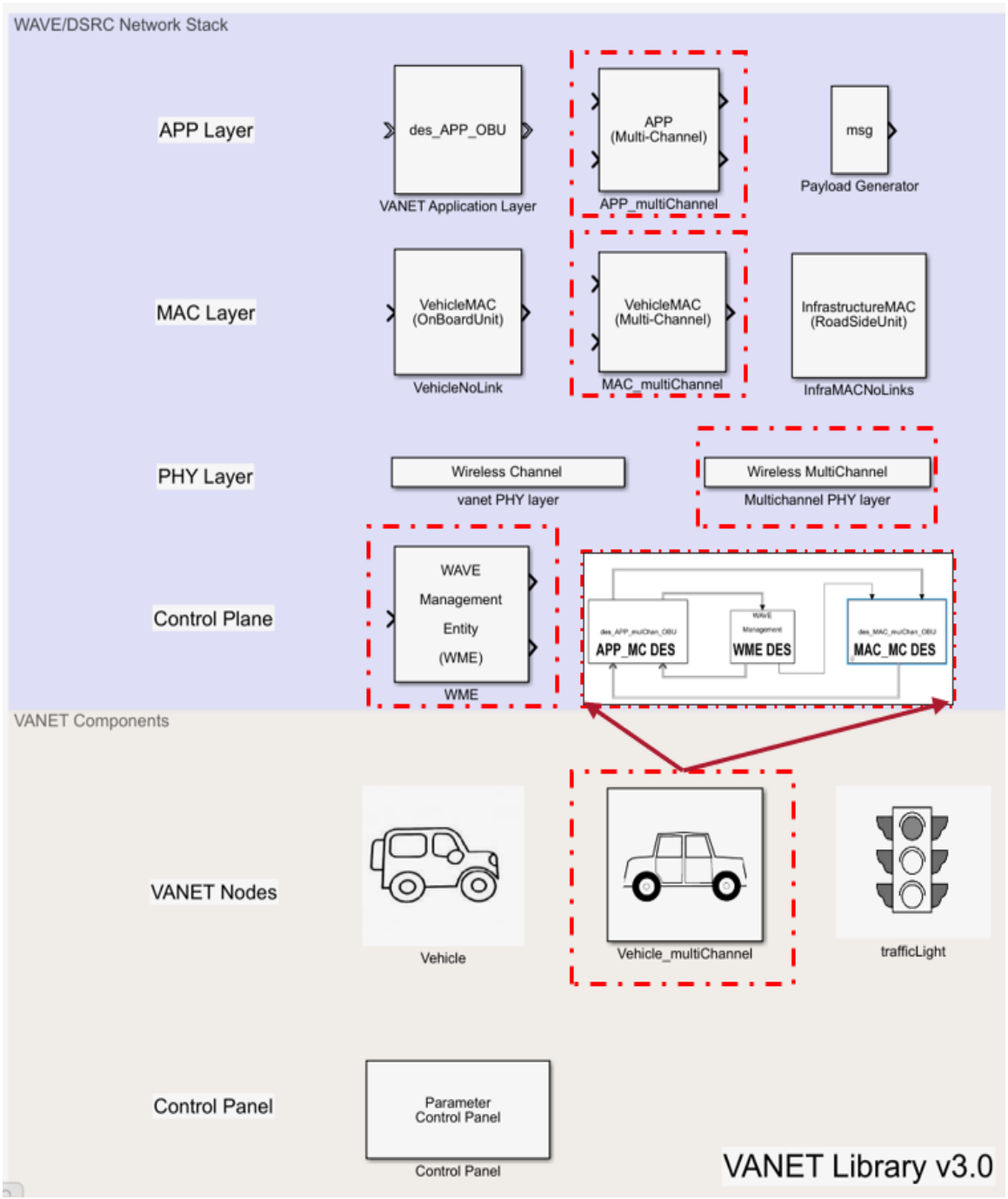


Figure 5.6: VANET Library V3.0 with multichannel blocks. The multichannel (MC) version of the APP DES, MAC DES and PHY DES modules are added to the library. The WME DES module is newly designed to process multichannel information.

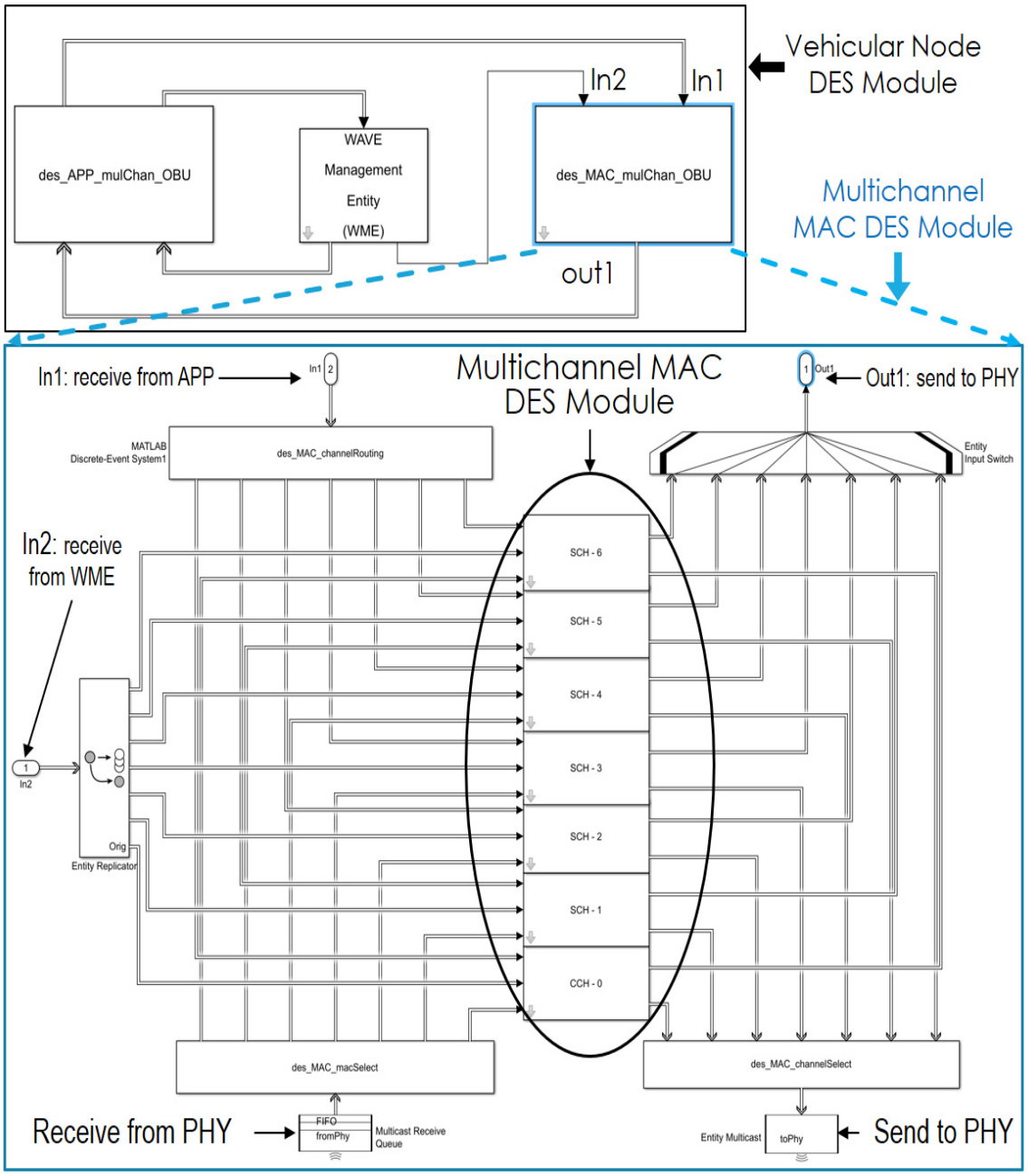


Figure 5.7: Full stack of multichannel vehicular network node and design details of multichannel MAC DES module. The PHY Tx and Rx functions are integrated with MAC DES modules, *i.e.,* not shown in the figure.

MAC DES module, and WAVE management entity (WME) module. The PHY Tx and Rx are integrated together with a multichannel MAC DES module. The details of multichannel MAC DES module are shown in the bottom portion of Fig. 5.7 (bottom), which has seven MAC submodules that correspond to one CCH MAC and six SCH MACs. The reason to have six SCH MACs is that a vehicular node may be involved in multiple non-safety services within a overlapping period. For example, a vehicular node can send messages for *Service 1* during SCHI/1, messages for *Service 2* during SCHI/2. At the end of SCHI/1, some of the Service 1 data is still in the backoff process. As a result, the activities are paused due to the end of SCHI/1 but the status of this data is retained inside SCH_MAC(1) until the next SCHI for Service 1, *e.g.,* SCHI/3, while the unsent data of Service 2 is saved inside SCH_MAC(2). The data from the APP layer enters the MAC module from the input port *In1* and is distributed to the corresponding MAC submodules based on the multichannel information saved in the *channel field*. In the MAC submodule, the data is converted into waveforms and finally broadcasted to the wireless channel. On the other hand, after receiving a waveform from the wireless channel, the MAC module extracts the payload and sends it to the APP layer via the output port *Out1*.

If a vehicular node is not starting nor joining any non-safety services, all the SCH MAC modules are deactivated with only the CCH MAC module being active during CCHI. For a vehicle node initiating a service, the vehicle becomes a service provider (SP). The APP DES module of the SP will first query the WME DES module about the appropriate SCH. Then, a WAVE service advertisement (WSA) message is created with the SCH information provided by the WME . This WSA will be sent into the wireless channel immediately if the SP is currently in CCHI or waiting to be sent in the next CCHI. Meanwhile, the WME sends control messages to the multichannel MAC DES module that have the chosen SCH MAC submodule in order to get ready for the next SCHI. When receiving a WSA, the vehicular nodes that are interested in the service become service users (SUs). The SUs extract the multichannel information from WSA and send to WME DES module, which logs the SCH information and inform the multichannel MAC module to activate the corresponding MAC sub-channel DES module for the next SCHI.

5.2.1 Interaction between APP layer and WME in DES

The intersection between the APP layer and the WME is the most complicated structure in multichannel operations since an internal *management* message transmission system is required for the multichannel information inquiry, reply, update, and coordination actions, as shown in Figure 5.8. Unlike the single channel APP layer, who just generates messages and sends them to the MAC layer, the multichannel APP layer needs to consider the type of messages and adds multichannel information to the message.

For safety-related messages, such as BSMs, or control messages, such as WSA, the channel number is fixed to *CH178*, *i.e.,* CCH. Non-safety services are referred to as *'activity'* in the code body of *sendMCmsg('activity')*. Two generation events with different tags, *activity* and *requestMCinfo*, are triggered. The former is the event to generate the non-safety service data. However, the SCH information is currently stored in the WME module and is unknown to the APP layer module, thus the non-safety message is unable to attach the SCH information with the entity. Consequently, a short delay, *MCinforDelay*, is set to delay the generation process in order to let another event *eventGenerate('requestMCinfo')* obtain the SCH information from the WME module.

The event *eventGenerate('requestMCinfo')* generates an internal management frame of type 2, *i.e.,* mgmFrame(2)., where mgmFrame(2) is forwarded out of the APP layer module via output port 2 and received by the WME module via input port. The *enter* activity triggers the action *mgmFrameEntry*, which selects an available SCH based on the SCH/AC information in the SCH vector by a self-defined method, *selectSCH()*. Then, another management frame, mgmFrame(4), is created containing the SCH information and sent back to the APP layer. The moment mgmFrame(4) enters the APP DES module, the action *mgmFrameEntry()* is triggered, in which the SCH information is saved by a class property (member variable), *obj.spSCHInfo*.

A WSA is immediately generated with the SCH information by the event *eventGenerate('WSAgen')*. Moreover, SCH information is also used by the action *payloadGenerate('activity')* after a delay of *MCinfoDelay* in order to create non-safety service messages. Both service message and WSA messages are sent to the multichannel MAC layer. Note

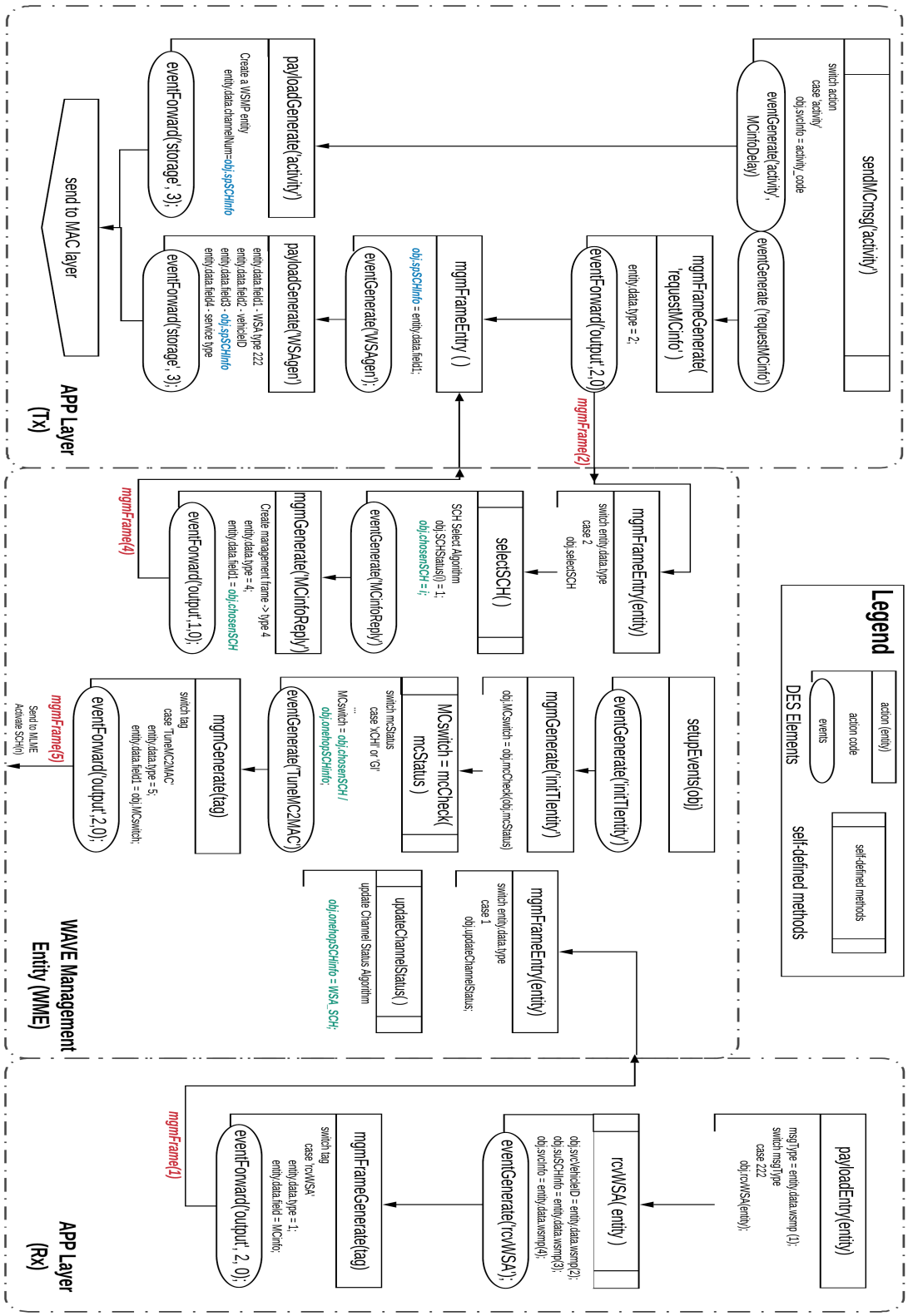


Figure 5.8: Model WBSS establishment with DES. Management frames are created between APP DES module and WME DES module to coordinate multichannel operations. WME module also uses timerevents( ) to alternate CCH/SCH intervals.

that the WSA is a CCH message while the 'activity' is a SCH message. The WSA will be transmitted immediately after the backoff if it is currently in the CCHI or in the next available CCHI if it is currently in the SCHI. The 'activity' message is always transmitted in the following SCHI after the WSA has been sent.

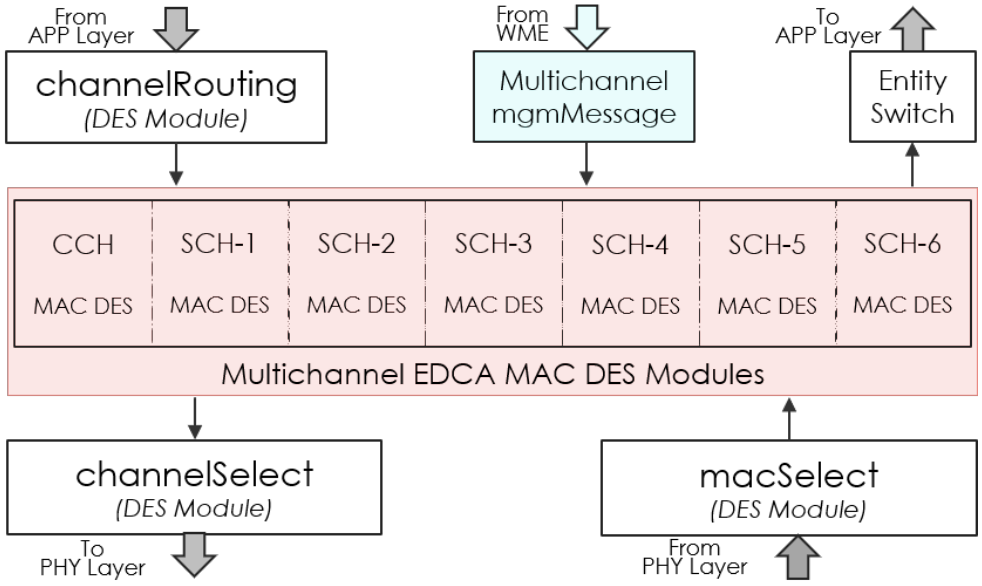
At the SU end, when a WSA arrives at the APP layer, the action 'payloadEntry' is triggered followed by a self-defined method 'rcvWSA', which creates *mgmFrame(1)* using the *eventGenerate(tag)* event and sends it to the WME of the SU. The WME updates the SCH information in the SCH vector and prepares for joining the WBSS in the next CCHI.

The WME module is also responsible for sending updated multichannel information to the MAC layer periodically, as shown in the central DES flow in the WME module of Figure 5.8. In the self-defined method, *MCswitch = mcCheck(mcStatus)* and the WME module maintains a series of timer events corresponding to the 4 ms GI, 46 ms CCHI, and 46 ms SCHI. These timer events are triggered reciprocally in a cycle whenever a channel interval is changed by the timer events. The up-to-date SCH information is sent to the multichannel MAC layer via *mgmFrame(5)* such that the multichannel MAC layer module is able to switch on the corresponding SCH_MAC submodule at the beginning of next SCHI. When the service is finished, the WME module will inform the multichannel MAC layer to switch back to the CCH at the start of the next CCHI.

5.2.2 Multichannel MAC/PHY Layer DES Module

Figure 5.9(a) shows the framework of a multichannel MAC/PHY layer DES module. The central multichannel EDCA MAC DES module consists of seven subchannel MAC DES objects, each of which inherits the same base class *des_MAC_mulChan_OBU*. The subchannel MAC modules are differentiated by the class property *obj.ChanNum*. The detailed design of the EDCA MAC layer is presented by the authors in [113, 117]; this paper will only focus on the MAC behavior related to multichannel operations.

The payload entity from the APP layer enters the *channelRouting* DES module, which dispatches it to one of the subchannel MAC modules according to the multichannel information of the payload. In the MAC layer, payload entities are converted into frame entities and the multichannel information is copied to the frame entity in the header field.



(a) Structure of multichannel MAC DES modules. Multichannel management frame (mgmMessages) activates the CCH DES during CCHI or at most 6 SCH modules during SCHI.

```
function [entity,events] = mgmFrameEntry(obj,~,entity,~)
    % mgmFrame(5) - entity.data.field1 = selectSCH.
    obj.SCHidx = entity.data.field1;
    if obj.SCHidx == obj.ChanNum
        obj.MCSwitch = 1;
    end
    events = obj.eventDestroy();
end

function isChannelValid = channelCheck(~, channelBusy, MCSwitch)
    if ~channelBusy && MCSwitch
        isChannelValid = 1;
    else
        isChannelValid = 0;
    end
end
```

(b) The DES code for multichannel MAC module switching operation. Two prerequisitions for an active channel: First, flag from WME module is 1, and Second, channel is idle. Otherwise, the MAC module is set to deactive status.

Figure 5.9: Design multichannel MAC layer with EDCA using DES. Partial code is provided to show how WME module alternates multichannel status to MAC layer.

The frame entity that is selected during the internal contention process is forwarded to the *channelSelect DES module*, which selects the corresponding PHY channel frequency based on the multichannel information in the header. On the other hand, when a waveform entity is received from the PHY link, the macSelect DES module records which PHY subchannel the waveform entity comes from. The PHY subchannel information corresponds to the designated subchannel MAC module, thus the waveform is forwarded to the specific subchannel MAC module, *i.e.,* CCH or SCH1-6, where the CRC is performed on the waveform and the payload is extracted in order to be sent to the APP layer via *Entity Switch* block.

Similar to the APP layer DES module, the MAC layer module also accepts the management frame (mgmFrame) from the WME DES module. Referring to Figure 5.8, we know that the management frame sent to the multichannel MAC layer is *mgmFrame(5)*. After mgmFrame(5) enters the MAC DES module, it triggers the *mgmFrameEntry* action, as shown in the Figure 5.9(b). The *field1* field contains the multichannel information obtained from the multichannel coordination process performed by the APP layer and the WME module. The multichannel information is compared with the *obj.ChanNum* property of each individual subchannel MAC layer module. When obj.ChanNum equals to the received channel index, the multichannel switch property, *obj.MCSwitch* is set to 1, which means the corresponding subchannel MAC module is activated until further multichannel information is received from the WME module.

In single channel mode, the backoff process in the MAC layer only needs to consider the channel status. If the channel is busy, the backoff process is paused until the channel becomes idle. On the other hand, the multichannel scenario needs to take both channel status and channel switching into consideration. During the CCHI, all SCH MAC modules are set to the busy channel status while in the SCHI the CCH MAC module is set to busy. During the GI, both the SCH and CCH MAC modules are set to busy status. The function *channelCheck* is created for this purpose, as shown in Figure 5.9(b). Whenever the backoff operation needs to query the channel status, it calls the channelCheck function, which checks both channel sensing result (channelBusy) and multichannel status (MCSwitch) together. Only when the channel is not busy ($channelBusy = 0$) and the multichannel switch is on (MCSwitch == 1) shall the backoff continue or else the backoff is paused.

In a vehicular network, a complete point-to-point physical (PHY) layer consists of a transmitter (Tx), a receiver (Rx), and a PHY link. When the MAC layer passes the frames to the PHY layer, the Physical Layer Convergence Protocol (PLCP) sublayer of the PHY Tx prepares the MAC protocol data units (MPDUs) for transmission as the PLCP Service Data Unit (PSDU). A PLCP header containing the information such as code rate (MCS), length of the data field is prepended to the PSDU. Then, a preamble is added to the PSDU for the purpose of synchronization and carrier frequency offset (CFO) correction between the Tx and the Rx. With the PLCP preamble and header, a PSDU is converted to a PLCP Protocol Data Unit (PPDU), *i.e.,* a waveform at the bit level. The whole Tx activity is implemented via the *psdu2waveform* function and it is integrated with the MAC DES module.

The waveform (PPDU) is sent to the wireless PHY link, which is a DES module to simulate air propagation delay. During the delay period, the waveform is passed through an Additive White Gaussian Noise (AWGN) channel, with the random noise being added to the waveform bits. More complex channel models can be selected by the PHY link DES such as the two-ray ground reflection model. In the PHY DES object, a series of persistent variables and map containers are used to buffer the transient channel status. For example, the *status* variable is used to record whether the channel is busy or not. A 1 $\times$ waveformLength vector, *waveformBuff*, is to buffer the waveform bits during the air propagation delay period. *txMAP* is a vector used to store the Tx addresses when a waveform enters the PHY link DES module. This is designed to deal with the retransmission situations for reliable data transmissions (RDTs), *i.e.,* Data-ACK transmission format.

At the PHY Rx side, when the waveform arrives at the Rx the preamble field is extracted for a series of wireless receiver operations including synchronization, coarse CFO correction, and fine CFO correction. With the MCS and length information from the header, the data field is demodulated and converted into the PSDU. A cyclic redundancy check (CRC) is performed on the PSDU. If the PSDU passes the CRC, the content of the payload is sent to the APP layer. For PSDUs that fail to pass the CRC, they are discarded by the event *eventDestroy()*. The activity of the Rx is implemented by the *waveform2psdu* function and is integrated with the MAC DES module as the Tx function.

The activities from both PHY Tx and Rx are not event-driven and are implemented using regular functions. No additional DES frameworks are needed for the PHY Tx and Rx. Consequently, the function of the PHY Tx, *psdu2waveform*, and the function of the PHY Rx, *waveform2psdu*, are integrated with the MAC DES module. The PHY link activities, such as waveform entry and waveform exit, are event driven and implemented by an independent DES. The operations mentioned above are for single channel conditions. For the multichannel scenario, the PHY Tx and Rx functions only need to perform minor changes in order to contain the multichannel numbers. The modification to the PHY link DES is relatively complex as we are trying to constrain the total number of DES frameworks in the simulation in order to avoid a significant increase in the execution time. As shown in Figure 5.10, the multichannel MAC DES module consists of multiple MAC DES submodules, each of which corresponds to a single MAC subchannel. This design considers the implementation complexity since a single MAC DES module is complicated enough to deal with a large amount of operations such as four EDCA queues, payload to frame conversion, frame backoff, retransmission, and frame to waveform conversion. Note that it is very difficult to design one DES for n multichannel MAC layer behaviors.

For the design of multichannel PHY links, we can employ n replicas of the single channel DES module. However, an increase in the number of DES module may decrease the execution speed because the overall computational cost in terms of events also increases. Thus, our approach on developing the simulator is trying to use as few DES modules as possible. For the multichannel MAC layer, we have no options but to use multiple DES modules. However, the PHY link DES module is mainly responsible for buffering the waveform entities for a period of delay before sending to the receivers. This activity does not require multiple DES modules for different PHY sub-links, an extended single PHY link DES module is sufficient for dealing with such conditions. In the multichannel scenario, suppose we have n channels, all of which use persistent variables mentioned above and are expanded. *status* variable is expanded to n $\times$ 1 array, *waveformBuff* is expanded to n $\times$ waveformLength, and the txMAP is also expanded to a *containers.MAP* with n key-value pairs, as shown in Figure 5.10.

Usually one PHY channel link corresponds to more than one pair of Tx and Rx, thus

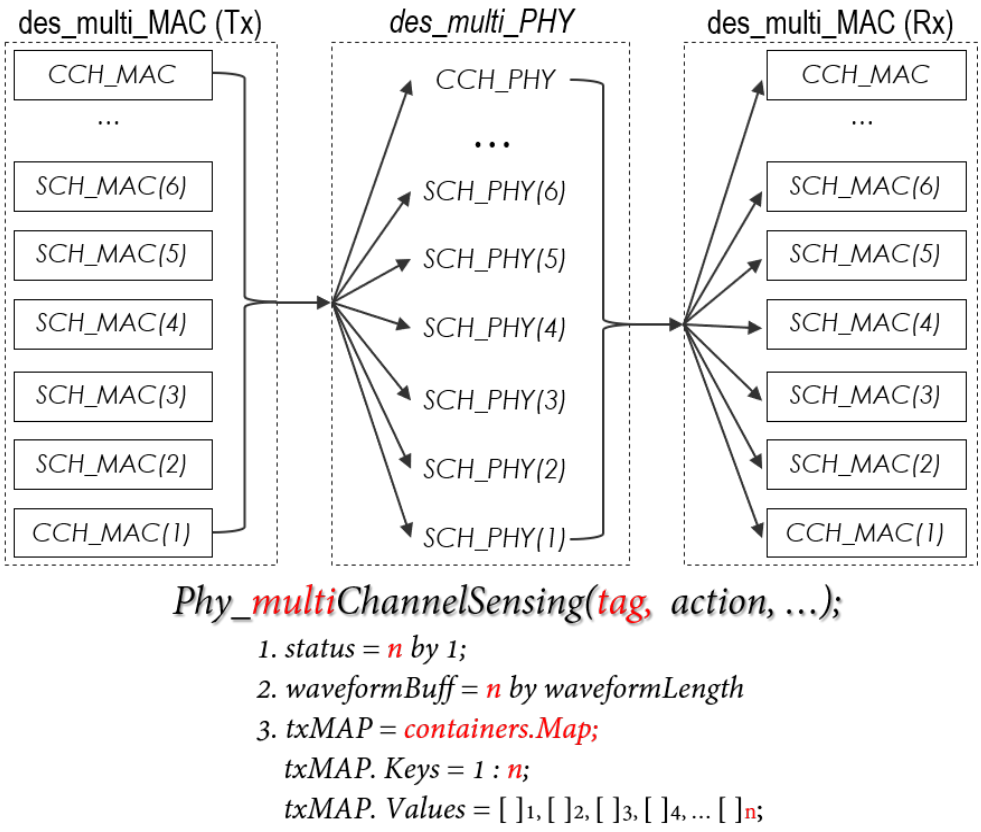


Figure 5.10: Implementation of multichannel PHY layer. The feature is using one DES module to mimic multichannel PHY link in consideration of execution speed.

a *Multicast* pair in DES is selected to deal with this *many-to-one* or *one-to-many* scenario (see the black arrows in Figure 5.10). A *Multicast* pair consists of one *Entity Multicast* module and one *Multicast Receive Queue* module. The *Multicast* pair allows more than one *Entity Multicast* modules to send entities to the same *Multicast Receive Queue*, or one *Entity Multicast* module sends entities to multiple *Multicast Receive Queue* modules as long as they share the same tag. In our design, waveform entities from different MAC modules, including multiple subchannel MAC modules within one vehicle or multiple MAC modules from several vehicles, are aggregated and transmitted through the single *Entity Multicast* module (tag: ToPHY), and then received by the *Multicast Receive Queue* (tag: ToPHY) of the PHY link DES, which dispatches the data flows to different persistent variable elements indicating PHY link buffers. After the air propagation delay, the PHY link DES aggregates

the waveform entity flows and send them to the *Entity Multcast* module (tag: FromPHY) and finally received by *Multicast Receive Queue* (tag: FromPHY), which connected to the Rx function in the MAC layer DES. Using a *Multicast* pair is effective when dealing with the condition of unequal number of Tx and Rx.

5.2.3 Test Case: Multichannel Lane Changing Activity

In order to validate the multichannel operation implementation we have introduced in the above sections, the lane changing behavior from section 4.2.3 is mitigated to the multichannel scenario. The coordination process of the lane changing activity in both single channel and multichannel scenarios is summarized below:

Lane changing behavior via single channel communication

- step 1: Lane changing service needed for car2.
- step 2: Car2 sends multicast lane changing requests to car3 and car4.
- step 3: Car3 and car4 send lane changing replies to car1.
- step 4: Car2 starts to change lane.

Lane changing behavior via multichannel communication

- step 1: Lane changing service needed for car2.
- step 2: Car2 creates and sends WSA frames via CCH during the CCHI.
- step 3: Car2,3,4 tune to the same SCH in the next SCHI.
- step 4: Car2 sends multicast lane changing requests to car3 and car4 during the SCHI.
- step 5: Car3 and car4 send lane changing replies to car1 during the SCHI.
- step 6: Car2 starts to change lane.

The above processes, the lane changing coordination messages are set as the *non-safety* messages in order to be exchanged during the SCHI. Thus the multichannel lane changing

behavior involves one more stage, *i.e,* the multichannel coordination to establish a WBSS. In addition, the multichannel lane changing process covers at least two channel intervals: one CCHI and one SCHI. The simulation result is shown in the Figure 5.11.

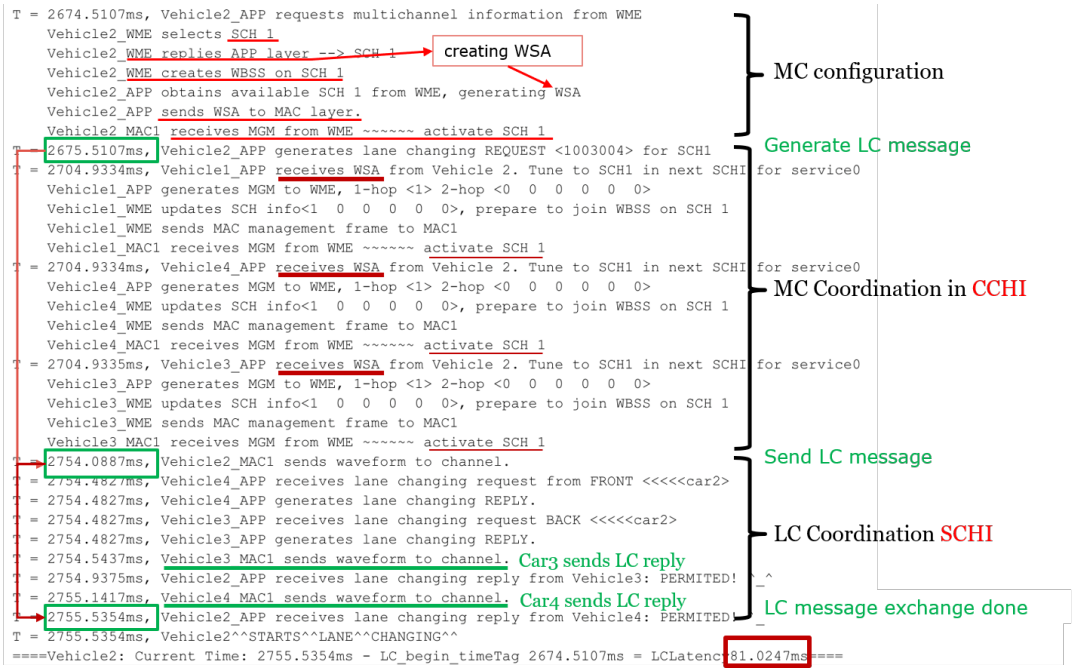


Figure 5.11: Coordinated lane changing behavior based on multichannel V2V communication. The coordination consists of two stages: WBSS establishment coordination and lane changing coordination.

In the figure, the lane changing (LC) request is generated during the CCHI and is configured to be sent during the next SCHI. Meanwhile, a WSA frame including the SCH channel information is created by car2 and sent over the CCHI. A LC WBSS is established with the WSA frame. Then the LC coordination starts in the adjacent SCHI. The overall service latency is around 81 ms, while the latency of lane changing operation over single channel communication is around 2 ms.

In single channel scenario, the latency mainly comes from the backoff delay of each LC message. In the multichannel scenario, the channel switching operation is the major source of the latency. The situation shown in Figure 5.11 is not the worst scenario, *i.e.,* the LC trigger time instant happens to be during the CCHI, thus the WSA as the control frame can be transmitted immediately. The worst scenario is the LC may be triggered during a

SCHI, then the WSA frame has to wait until the next CCHI to be transmitted. The whole LC process in the worst scenario may cover three channel intervals. Therefore, the latency might be much longer than the results shown in the figure.

The above analysis infers that setting the LC messages as non-safety type to transmit during the SCHI will introduce extra latency cause by the channel switching operation. In order to decrease the latency between the multichannel coordination and lane changing coordination process, the LC messages should be configured to safety-related property and transmitted during the CCHI.

5.3 Evaluation of Multichannel Vehicular Network

The primary motivation of vehicular networks is to improve traffic safety. Meanwhile, non-safety applications are also allowed to transmit via vehicular networks. In the single channel situation, the safety-related and non-safety applications are mixed together and share the only channel, with the performance of safety-related services potentially being affected. Even though EDCA is applied in order to classify the critical levels of different services, when the network traffic load increases, it may exceed the ability of EDCA to guarantee quality of safety-related services.

On the other hand, the multichannel separate the safety-related applications with the non-safety applications. The CCH is only used for safety-related messages in order to guarantee low latency communications while the six SCHs are assigned for non-safety applications requiring high throughput. However, a decrease in channel interval duration (46 ms) increases the contention probability, which may cause additional latency and a higher packet drop rate. Additionally, the control frames (WSA and WTA) for channel coordination create overhead data flows in CCH, and these data flows may interfere with safety-related applications. Furthermore, for every 100 ms each service is only allowed to use 46% of the channel interval. If a service is unable to finish in its channel interval, it has to wait for at least 54 ms to continue transmitting. Thus, it is difficult for a multichannel approach to improve upon single channel communications. In this section, we designed a series of simulations to evaluate the performance of the safety-related applications when

they coexist with non-safety applications in both single channel and multichannel scenarios.

5.3.1 Performance of Multichannel Operations with Mixed Services

The computer simulations involve up to 30 vehicles operating in a highway scenario. The wireless channel of the PHY layer consists of a two-ray Ground Reflection Channel model and an Additive White Gaussian Noise (AWGN) model, which are based on line-of-sight (LOS) conditions specified in [86]. The EDCA in the MAC layer use parameters from Table 5.2. The APP layer creates services consisting of n data flows per AC. Meanwhile, all vehicles are broadcasting BSMs (AC2) at 10 times per second.

The simulations are classified into single channel (SiCH) scenarios and multichannel (MuCH) scenarios. In the SiCH scenario, the non-safety messages are mixed with the safety-related messages but with different priorities. In the MuCH scenario, the non-safety service data are shared among n SCHs, and only the safety-related messages (AC2-AC3) along with the control messages (AC1) such as WSAs are transmitted in the CCH. Since the standard [80] does not specify the criteria on how to choose the SCHs for the different WBSS, we developed our own SCH reservation scheme, which reduces the probability of the hidden terminal problem by exploiting the SCH information from the two-hop vehicular nodes (details of our proposed SCH reservation scheme will be provided in Section 5.3.4). Other simulation parameters are shown in Table 5.2.

Table 5.2: Simulation Parameters for all simulations.

Parameter	Value
Modulation	BPSK
Data size	100 bytes
Data rate	3Mbps
Number of vehicles	30
Length of highway	1600 meters

In this work, we evaluated the performance of the safety messages and non-safety messages in both SiCH and MuCH scenarios. As the safety-related services require a high packet

delivery rate (PDR) and low latency, we compare the PDR and latency for safety-related messages between the SiCH and MuCH scenarios. The non-safety services require high throughput, thus we compare the throughput and latency for non-safety messages between SiCH and MuCH scenarios.

Packet Delivery Latency

The data latency is measured for each successfully delivered waveform. The latency is defined from the time instant when the data is created in the APP layer until the time when this data is delivered to the receivers. In this paper, all data transmissions are broadcasts, *i.e.,* no retransmissions. Consequently, the latency can be expressed as Eq. (5.4).

$$L_{all} = L_q + L_{bf} + L_{tx} + L_{airprop}. \tag{5.4}$$

Note that L_{all} indicates the overall period of latency the data may experience, L_q is the delay when a data is waiting in an EDCA queue before starting the backoff process, L_{bf} is the delay caused by the backoff process, L_bf may be large due to heavy traffic load or small contention window, L_{tx} is decided by data length and sending rate, while $L_{airprop}$ is the air propagation delay decided by the distance between the sender and receiver. L_q and L_{bf} are the main sources of overall latency.

Packet Delivery Ratio (PDR)

During the transmission, the packets may experience packet collision or interference due to other factors. These factors may cause bit errors in the waveforms, and if the error is too severe to correct the information contained might not be interpreted correctly. When a packet is received by a Rx, it has to pass the CRC otherwise it is regarded as corrupted and it will be discarded. In our experiment, the PDR is calculated separately for each AC and it is used to evaluate the channel congestion. Suppose the number of vehicles in the experiment is equal to $N_{vehicle}$, with the number of sent messages per vehicle per AC being equal to N_{Tx} and the number of successfully received packets for all vehicles being equal to N_{Rxall}, then the PDR can be calculated by Eq. (5.5).

$$PDR = \frac{N_{Rxall}}{N_{Tx} \times N_{vehicle} \times (N_{vehicle} - 1)}. \tag{5.5}$$

The value $N_{vehicle} - 1$ is due to the fact that each vehicle broadcasts to all the other vehicles except itself, thus the PDR should eliminate the amount of packets each sender received from itself.

Network Throughput (TP)

The network throughput is designed to measure how many data units a network can process within a specific time period. It is calculated by the amount of data moved successfully from the Tx to the Rx in a given period, as shown in Eq. (5.6).

$$TP = \frac{N_{Rxall}}{T}. \tag{5.6}$$

Note that N_{Rxall} indicates the total number of successfully received packets and T is the simulation period. In order to perform tests in extreme scenarios, we fix the total number of transmitted packets and enable burst transmissions for all vehicles. Each vehicle is required to transmit a fixed amount of packets per AC. Once the current packet is successfully sent to the receiver, the next packet is entering backoff process. The packets are sent one by one in this pattern until all packets are sent, after which point the simulation is ended.

Fig. 5.12 shows the trend of average data delivery latency for all ACs of SiCH and MuCH scenarios. In the simulation, each vehicle is required to broadcast 5 to 30 service messages. As the amount of broadcast messages increases, the overall latency increases for all ACs. This figure confirms that EDCA works as expected because the higher priority data enjoys much less latency relative to lower priority data for both single channel and multichannel scenarios. Note there is a crossing between single channel AC3 (S3) and multichannel AC3 (M3) at around 17 data flows. The threshold indicates that when the communication traffic load is low, the latency caused by the multichannel coordination occupies a larger proportion. When the network traffic load is high, more data can be transmitted during a CCHI/SCHI, thus the average latency increases but slowly. However, the single channel latency increases significantly due to L_q. The latency curve of MuCH/AC0 (M0) is almost

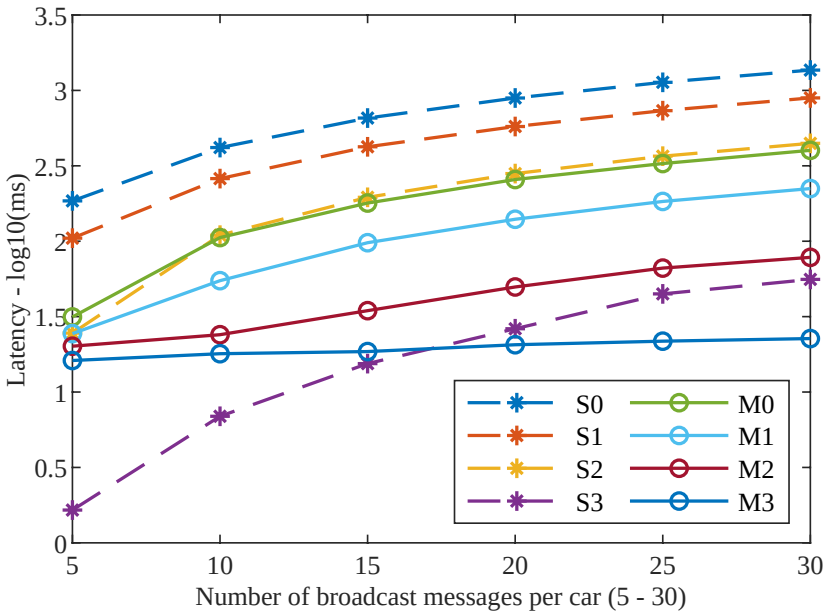


Figure 5.12: Average latency trend for single channel AC0 - AC3 (S0 - S3) and multichannel AC0 - AC3 (M0 - M3). The data flows per service increase from 5/service to 30/service. The number of vehicles is 30.

overlapping with SiCH/AC2 (S2), which indicates the lowest priority (AC0) in multichannel scenario performs similar with the second highest priority (AC2) in single channel scenario.

Fig. 5.13 shows a boxplot of latency for single channel communication (SiCH), Single channel communication with BSM Purge (SiCHwBP) and Multichannel communication (MuCH) of 30 vehicles with 30 data flows per service. For SiCH scenario, data with the lower priorities (AC0-2) experience longer latency than AC3 data. With heavy data traffic, the single channel is saturated, the unsent data are accumulated in the AC queues causing L_q to grow significantly. In particular, lower priority data has to give way to higher priority data due to EDCA, with L_q for lower priority data being more serious. J2735 standards [74] specify that the maximum latency of safety-related messages should be lower than 100 ms. We observe in Fig. 5.13 that the single channel AC0-2 cannot fulfill this requirements since the latency is larger than 100 ms. When all 4 ACs are enabled in the single channel scenario, only AC3 has latency lower than 100 ms and can be used for safety-related services. The performance of the single channel transmission with BSM purge feature is also observed

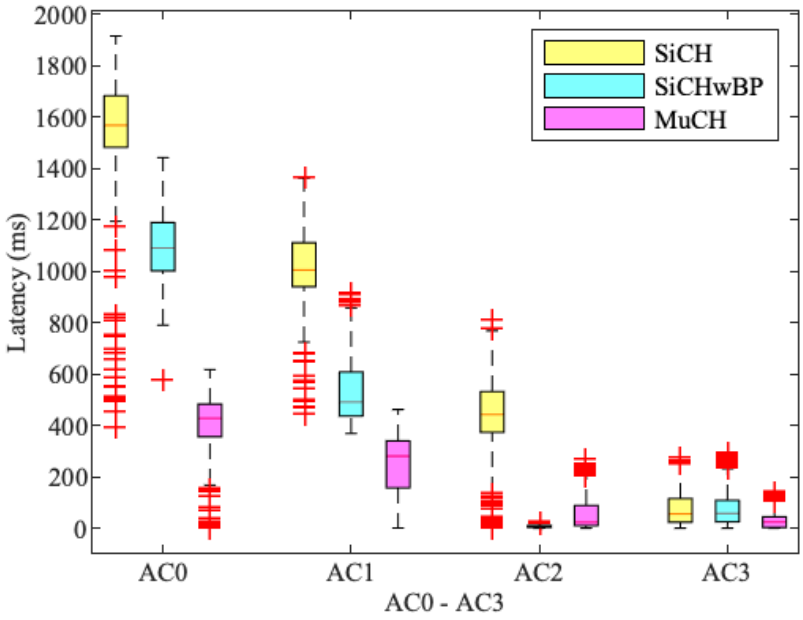


Figure 5.13: Boxplots of packet delivery latency for data from AC0 to AC3 under single channel condition, multichannel condition as well as multichannel with BSM purge.

from Fig. 5.13 (SiCHwBP). Research from [129] shows that a vehicle is forced to drop over 80% of messages due to no channel access before the next beacon message is generated with CSMA/CA scheme. In our simulation, we enable four EDCA AC queues and only AC2 is transmitting beacon messages (BSMs). When a BSM enters the AC2 queue, it purges all unsent message ahead of it. In this scenario, the BSM maintains a relative low latency as it clears all the other unsent data in the AC2 queue so that the L_q is significantly reduced with the cost of severe packet drop rate on AC2. Since a large amount of AC2 messages are purged by BSMs, the overall amount of transmitted messages is decreased accordingly, thus SiCHwBP experiences a lower latency than SiCH scenario. However, the discarded data due to the purge action can not be ignored. In SiCH, higher priorities AC2/3 should be used by safety messages exclusively in order to guarantee their lower latency.

For multichannel transmission, even though the reduction of the contention window as well as the multichannel coordination may induce extra latency, when the number of data flows is large, *e.g.,* 30 data flows $\times$ 30 vehicles, the MuCH has a lower latency than SiCHwBP on all ACs except AC2. SiCHwBP/AC2 achieves the lowest latency by flushing all unsent

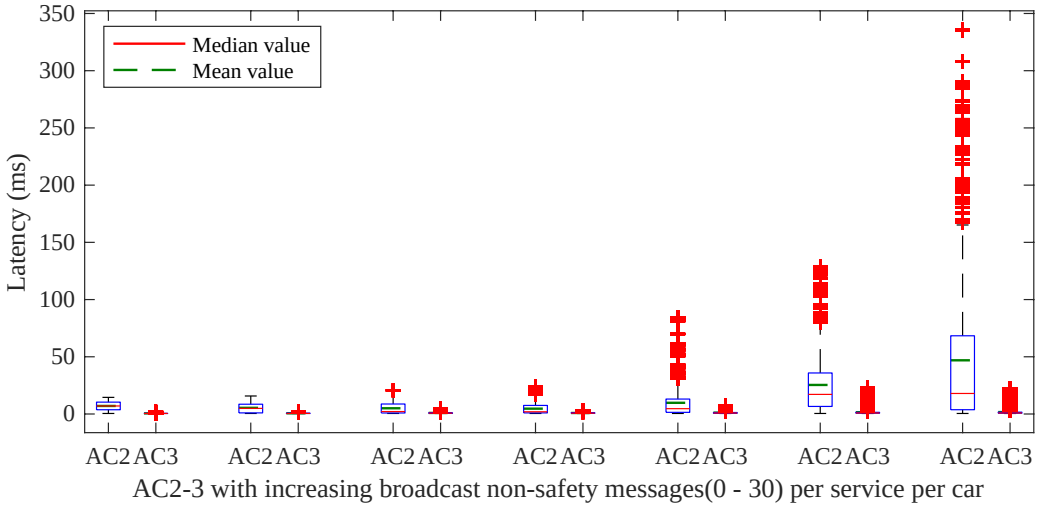


Figure 5.14: Impact to safety-related messages (AC2-3) in single channel mode. Non-safety messages (AC0-1) cause increment of average latency to safety-related messages and many long latency outliers.

data while MuCH/AC2 does not clear any unsent data. In MuCH scenario, we may assign AC3 to emergency safety messages, AC2 to BSMs and AC0-1 to control messages during the same CCHI. Besides, all four ACs can be used by non-safety services during the same SCHI for higher throughput purpose.

5.3.2 Performance of Safety-related Services

The first set of experiments explored the impact to safety-related messages in terms of latency when both safety-related and non-safety services coexist in the single channel environment. We defined four services corresponding to four ACs: two non-safety services (AC0 and AC1), one BSM service (AC2) and one critical emergency (EMG) service (AC3). The EMG service messages are generated using a Poisson distribution with $\lambda = 2$. BSMs are beacon messages with a generation interval of 100 ms. The vehicles are grouped into WBSS to share non-safety messages. The WBSS is formed on a Poisson distribution with $\lambda = 2$. Once a WBSS is established, the members start to communicate non-safety messages in consecutive bursts. We define n as the number of data flows each vehicle should broadcast before the service is finished. In the simulation, the number of vehicles is fixed at 30 and n

is increased from 5 to 30. The packet delivery latency of the successfully delivered packets are collected and the average latency of the safety-related messages, *i.e.,* BSMs (AC2) and EMG (AC3), are shown in Figure 5.14.

In this figure, the AC2 and AC3 data are shown as a group of boxplots. The first group is the reference group with no non-safety messages transmitted, *i.e.,* only AC2 and AC3 messages are sent. This is the best scenario for safety-related services since no interference comes from the non-safety services. Furthermore, we observe that AC3, the top priority, enjoys the benefits of EDCA and maintains at a relative steady and concentrated average latency. The mean and median values of both AC2 and AC3 data are consistent. However, with the addition of non-safety messages, an increasing amount of outliers start to appear for both AC2 and AC3 service messages. The average latency of the BSMs (AC2) increases from around 10 ms to 50 ms while the median latency only increases from 10 ms to 20 ms, which shows the latency distribution spreads to the higher value significantly.

At 30 non-safety message flows, the outliers begin to show that the AC2 latency reaches 350 ms. Reference [91] defines the latency of the BSM should not exceed 100 ms in order to ensure the effectiveness of the safety traffic information. Note that the EDCA is designed to enable the data with the higher priority enjoy less latency by sacrificing the performance of the data with lower priorities. This means the impact to the safety messages (AC2-3) coming from the non-safety messages (AC0-1) have already been alleviated by the EDCA. This result shows that single channel communication is unable to provide an efficient transmission environment for a mixed safety and non-safety services.

The latency of safety-related messages in a multichannel scenario is evaluated. Similar to the single channel experiments mentioned above, BSMs are broadcast at 10 Hz with AC2 priority and EMG messages are sent at a Poisson distribution with $\lambda = 2$. WBSS are established in order to share non-safety messages during the SCH intervals. Figure 5.15 shows the latency of safety-related messages in a multichannel scenario. Note that the non-safety messages in a multichannel scenario are migrated to the SCHs and are separated with the safety-related messages, with the traffic patterns of non-safety messages causing no direct affects to the safety-related messages, *i.e.,* no matter how many non-safety messages are transmitted within in each WBSS the safety-related messages behave almost the same.

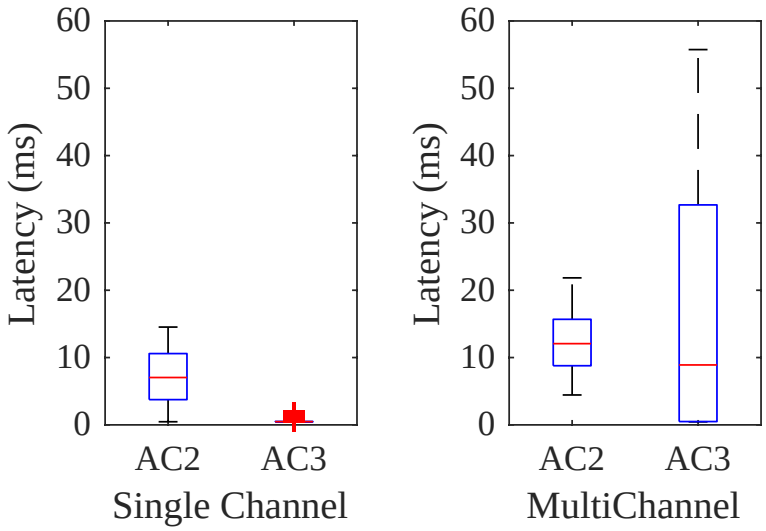


Figure 5.15: Latency of safety messages in multichannel mode. Avg. latency (AC2) is slightly increased due to the intense contention window. Random safety-critical messages are affected by channel switching operation.

Thus, we only extract one group of latency values for safety-related messages and compare them with the same reference group in Figure 5.14. The reference group is the best case scenario in a single channel environment, *i.e.,* the safety-related messages use the whole channel without contention with non-safety messages.

In the figure, the distribution of AC2 latency in a multichannel environment is close to the latency in a single channel scenario. The overall latency increases by less than 10 ms due to two factors. First, the transmission opportunity of the safety-related messages are shortened to 46 ms CCHI out of 100 ms SI. Second, the overhead transmissions of the control messages, such as the WSAs in the CCH, are increased. In a multichannel environment, the communications of non-safety services require the establishment of a WBSS. At lease one WSA per WBSS is required, thus with additional WBSS formed, more WSAs are transmitted along with the safety-related messages in the CCH. This can also be considered as a disadvantage with respect to transmission opportunity since safety-related messages need to contend with control messages for channel access. The constraint channel access period increases channel contention, thereby causes longer backoff periods and results in relatively larger average latency.

When the non-safety message flows are more than 15, the BSMs (AC2) starts to experience relatively larger latency with a significant amount of outliers as shown in Figure 5.14. However, in a multichannel environment, the increase in non-safety message flows per WBSS do not show any impact on the latency of AC2 BSMs. The latency distribution of AC2 is maintained within a steady range. Reference [91] defines the latency of BSMs should not exceed 100 ms and multichannel BSMs fulfills the requirements. Thus, we can conclude that even though the overall latency of AC2 BSMs increases but still within the required range. With the heavy traffic scenario of non-safety services, the multichannel environment is effective in order to ensure the efficiency of AC2 BSMs.

On the other hand, the EMG messages (AC3) are a significantly affected relative to the single channel scenario. The median latency is below 10 ms and most of the AC3 data experiences a latency below 30 ms. It is worthing noting that some of AC3 data experiences much larger latency of more than 50 ms. This is due to the channel switching characteristic of the multichannel environment, *i.e.,* a safety message may be generated at the end of a CCHI or during a SCHI, this safety message has to wait inside the AC queue until the next CCHI. Thus, a latency of at most 54 ms (50 ms SCHI + 4 ms GI) is added to the overall latency of this safety message. The reason why BSMs do not have this problem is because the timer from each car are synchronized with either a GPS or a WTA, thus the creation of the BSMs can be carefully controlled to make sure each BSM can be transmitted during a CCHI. However, the safety-critical messages are triggered by an emergency event, *i.e,* may be created at any time instance. Since the 50 ms delay is inevitable in multichannel communications, an EMG AC3 is likely to experience an additional 54 ms of latency. On the other hand, the average latency of AC3 is still below 30 ms and the higher latency situation of AC3 can be compensated for by other assisting technologies such as cameras with vision detection, radar, or lidar. Despite the good performance of AC3 in the single channel environment with respect to a smaller latency, the other three ACs are facing significantly larger latency, which makes them insufficient to provide a lower latency environment for the time-sensitive safety-related messages nor a high throughput environment for the non-safety messages.

The packet delivery rate (PDR) of the safety related messages were also evaluated and

the results are shown in Figure 5.16. Three sets of experiments were performed. First, we test the PDR when only safety-related messages (SFMs) are transmitted in the single channel (SiCH) scenario labeled as SFM(only). Since there are no non-safety messages, this is the best scenario for SFMs in a single channel environment and it is used as a reference. Second, we added non-safety messages to the single channel environment with lower priorities (AC0-AC1), with the amount of non-safety message flows fixed at 30 per AC, *i.e.,* each vehicle is required to broadcast 30 non-safety messages with a single AC to finish a service. Third, we repeat the simulation in the multichannel (MuCH) environment with the same configuration as with the second scenario. Choosing SFM (only) on SiCH as the reference group, it is not possible to tune the number of non-safety messages as was done for the previous two experiments. Thus, we fixed the number of non-safety messages to 30 for SFMs(mix) on SiCH and SFMs on MuCH scenarios, and increased the number of vehicles from 5 to 30. The simulations were repeated 10 times, and we calculated the PDR with a 95% confident interval (CI) with the results shown in Figure 5.16.

In the figure, when the SFMs are mixed with non-safety messages in a single channel, the PDR drops sharply from 90% (15 vehicles) to less than 50% (30 vehicles). Even though non-safety messages are at a lower priority, when the overall amount increases, *e.g.,* 30 msgs $\times 2$ ACs $\times 30$ vehicles $\times (30-1)$ broadcast receivers $= 52200$ burst msgs, the impact to the SFMs is significant. In this situation, EDCA is insufficient to guarantee the QoS of SFMs.

The multichannel SFMs perform similarly to the SFMs (only) scenario in SiCH. This means the multichannel SFMs are not affected by the increase in the number of vehicles and their non-safety messages. The major sources for dopped packets in the MuCH environment can be classified in two types. The first type is packet collision, which is the same with SiCH. The second cause is due to the channel switching, which is a special case for the MuCH communication. When the channel begins to switch to SCH frequency, it is possible that a Tx is in the middle of a transmission or a Rx is in the middle of receiving. In such cases, the transmissions are immediately canceled. On the receiver side, the incomplete packets are dropped. However, the SFMs are usually time sensitive, with retransmitting after 54 ms not being possible. Thus, these SFMs are often discarded by the Tx. The similar PDR values for the *SFMs on the MuCH* and *SFMs (only) on the SiCH* prove that

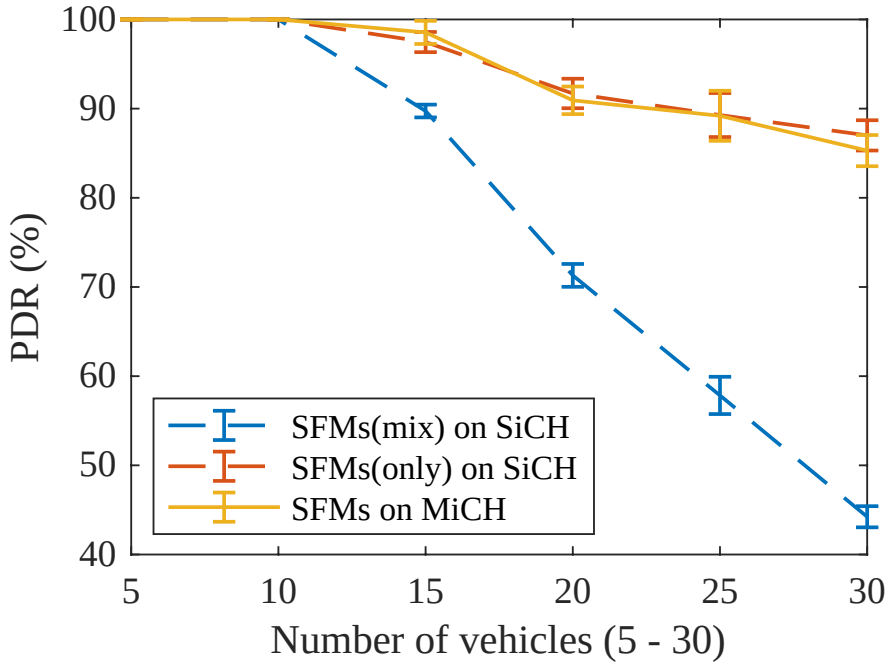


Figure 5.16: Packet delivery rate (PDR) comparison between single channel and multi-channel operations. Safety services in multichannel has a much higher PDR than in single channel when mixed with non-safety services and equivalent PDR with pure safety services in single channel mode.

the major factor causing significant packet drop is the non-safety transmissions. Since the non-safety transmissions are moved to the SCHs, factors such as channel switching and overhead transmissions of control messages (AC1) only cause minor impacts, which can be ignored. It is worth pointing out the reason we assigned the AC1 priority to the WSAs. It turns out that the WSAs serve non-safety services and should not have a higher priority relative to SFMs. Based on the analysis above, we can see that the multichannel operations do have several side effects with respect to the safety-related services including increased contention probability and overhead transmissions caused by control messages. However, when involving a high density of non-safety transmissions, multichannel operations are necessary to maintain the PDR of safety-related services.

5.3.3 Performance of Non-Safety Services

In this section, we focus our analysis on non-safety services. The first set of experiments are conducted to evaluate the latency of non-safety services in single channel (SiCH) and multichannel (MuCH) scenarios. In SiCH, AC0-AC1 are assigned to the non-safety services while AC2-AC3 are assigned to the safety-related messages. Consequently, only AC0-AC1 data is extracted and the latency is shown in the plot. For the MuCH scenario, the experiments are divided into two groups. First, only AC0-AC1 are used for the SCHs and AC2-AC3 are idle. This is to show the direct comparison on latency when non-safety services have the same traffic patterns in both SiCH and MuCH environments. Second, since the safety-related services are separated to the CCH, the AC2-AC3 can be assigned to the non-safety services in the SCHs, *i.e.,* the non-safety services can use all four ACs in the SCHs instead of only two in SiCH. The latency of non-safety services with four ACs enabled is also evaluated and compared with the SiCH condition. In the experiments, the total number of vehicles is fixed at 30 and the number of non-safety (NS) messages (MSGs) increases from 5 to 30.

The top portion of Figure 5.17 shows the average latency of non-safety services with AC0 and AC1 in SiCH and MuCH scenarios. The increment rate of MuCH is slower relative to SiCH. When the number of NS MSGs is 10 per vehicle, the average latency of SiCH AC0 (23.6ms) is lower than MuCH (29.1ms). After this point, MuCH AC0 maintains a lower

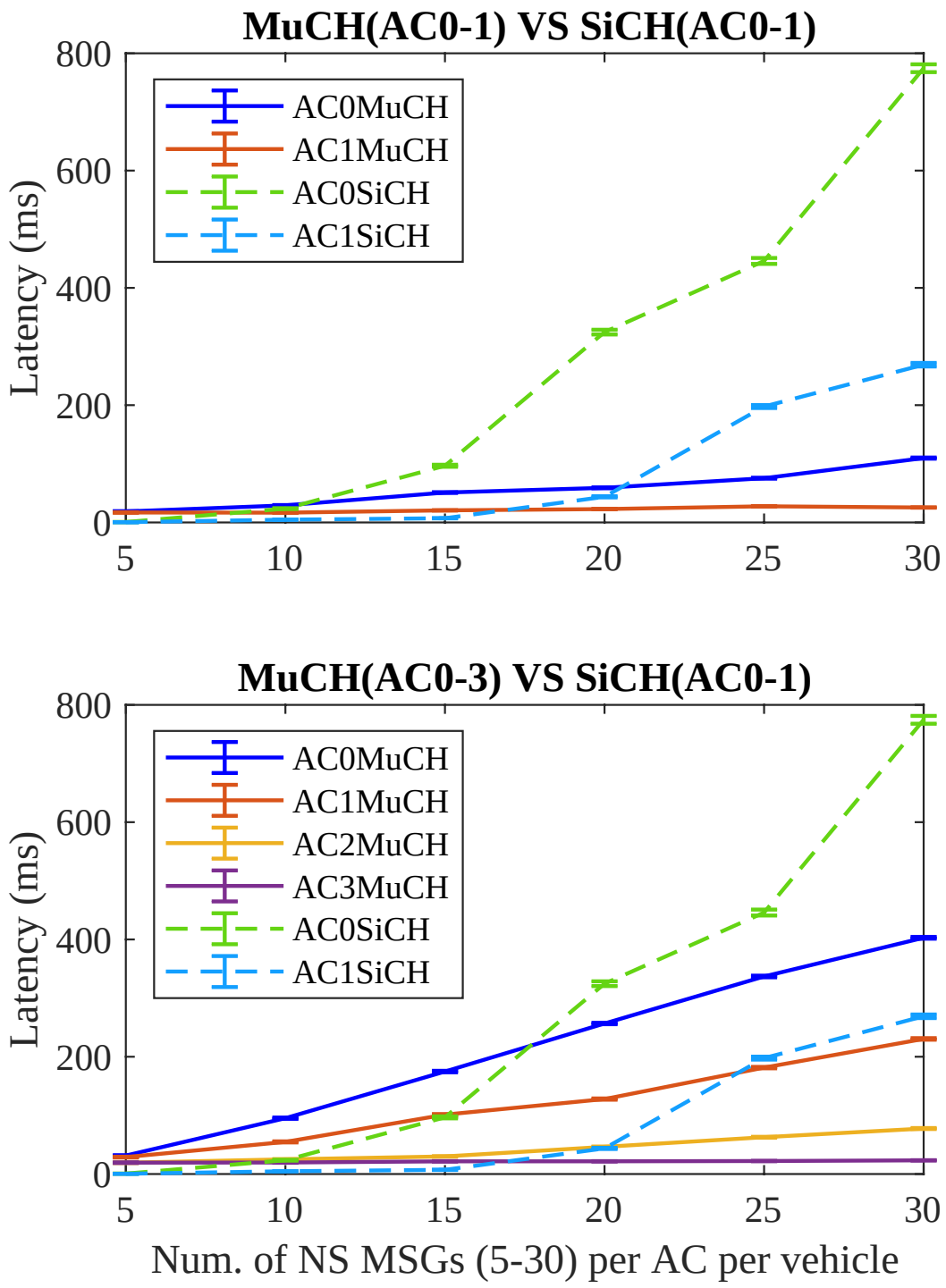


Figure 5.17: Latency comparison on non-safety services. With high density transmissions, single channel latency increases dramatically, while multichannel maintains a steady low latency. All 4 ACs are enabled in multichannel mode in contrast with 2 ACs in single channel.

latency than SiCH AC0. With 30 NF MSGs, SiCH AC0 experiences an average latency of 774.6ms while MuCH AC0 only has 109.7 ms. Similarly, the crossing point of AC1 appears at around 18 NS MSGs. The latency of SiCH AC1 ranges from 0.47 to 269 ms while MuCH AC1 ranges from 16.9818 ms to 25.47 ms. Thus, we can assert that when the transmission traffic is small, SiCH is more efficient because the time cost of the channel switching in MuCH is non-negligible. When the transmission traffic is heavy, the queue delay of the SiCH outweighs the channel switching delay, resulting in MuCH being more efficient under heavy traffic condition.

The bottom portion of Figure 5.17 compares the latency between SiCH (AC0-AC1) with MuCH(AC0-AC3). We can observe that the participation of AC2-AC3 increases the latency of AC0-AC1 in MuCH. As more NS MSGs are broadcast, SiCH latency increases exponentially while MuCH latency increases almost linearly. At the point of 30 NS MSGs, the average latency from AC0 to AC3 is around 402.87, 230.4, 77.6, and 23.3 ms, respectively. Compared with SiCH, having 774.6 ms on AC0 and 269 ms on AC1 shows that MuCH is able to ensure a lower latency than SiCH, especially in heavy traffic scenarios. On the other hand, only AC0 and AC1 can be assigned to non-safety services in SiCH in order to guarantee the QoS of safety-related services, while MuCH can enable four ACs and maintain a relative low latency. Therefore, we have reason to speculate that MuCH should be able to bring high throughput communications to non-safety services.

The network throughput (TP) is defined as the amount of data successfully delivered during a time period. The TP in SCHs is relatively complex as it is highly dependent on the total number of established WBSSs and the SCH reservation mechanisms. For example, if the vehicle density is low and all the vehicles are located within one WBSS, then only one SCH is utilized during one SCHI while leaving the remaining five SCHs vacant, *i.e.,* the channel is not fully used. Another scenario is related to the SCH reservation scheme, where multiple WBSSs reserve the same SCH. The channel contention within one SCH causes longer delay and lower packet delivery rate (PDR), thereby result in a lower TP. On the other hand, the idle SCHs result in wasted channel sources, with the major feature of multiple SCHs on TP not being fully used. In our case, we use the proposed cooperative SCH reservation mechanism, which fully considers SCH/AC information conveyed by the

WSAs to achieve a higher utilization on SCHs.

In our experiments, we increase the number of vehicles from 5 to 30 while also increasing the number of non-safety messages (NSMs) per vehicle in a WBSS from 5 to 30, *i.e.,* each vehicle is required to broadcast 30 messages to finish a non-safety service after establishing or joining a WBSS during a SCHI. If the vehicles cannot send all required NSMs before the end of the SCHI, the next SCHI will be reserved to continue the transmission. We calculated the TP for each situation and the results are shown in Figure 5.18.

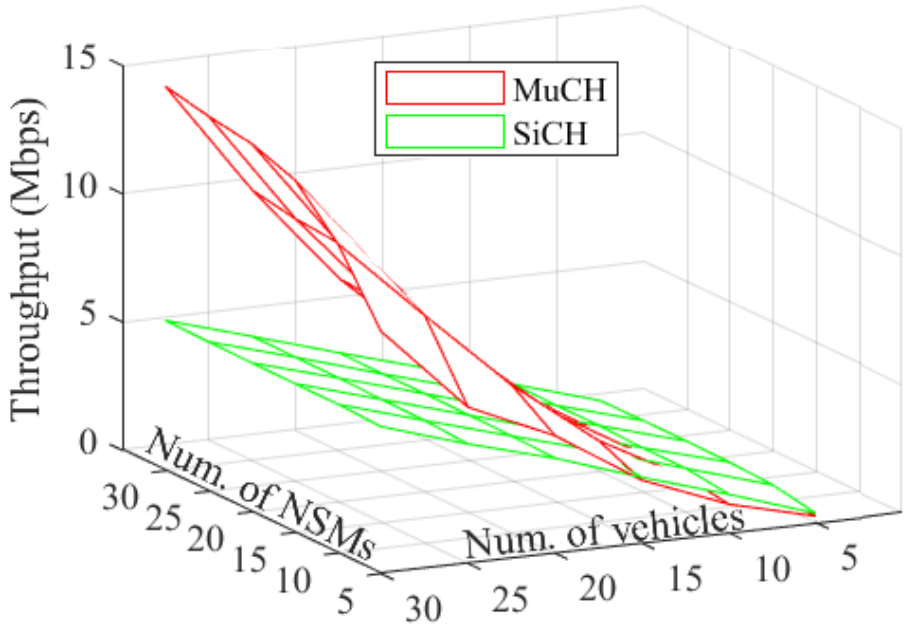


Figure 5.18: Throughput comparison between single channel and multichannel operations. Multichannel is effective for heavy traffic scenarios with much higher throughput.

When the total number of vehicles is fixed, an increase in the number of NSMs will result in a single channel (SiCH) scenario that maintains a constant TP of around 6 Mbps for 30 vehicles while the multichannel (MuCH) environment increases its TP between 8 Mbps to 16 Mbps for 30 vehicles. This proves that for the 30 vehicles situation that the SiCH is saturated at 6 Mbp while the MuCH is still able to increase the TP. For 30 NSMs in the 30 vehicles scenario, the MuCH TP is almost three times higher than SiCH TP. If

the number of NSMs is fixed, when the number of vehicle nodes increases, both SiCH and MuCH scenarios increase their TP almost linearly. For less than 10 - 15 vehicles, SiCH has slightly higher TP than MuCH scenario. This is because the number of WBSS is low, the multichannel SCHs are not fully utilized and the overhead on multichannel switching decreases the TP. After this crossover, additional vacant SCHs are utilized, MuCH starts to show its high TP feature. Conceptually, we may expect a n times increase in TP when using n SCHs relative to one single channel. However, the TP of the MuCH scenario is constrained by the 46% channel access period, MuCH communication can only achieve 3 times higher TP than SiCH communication for 30 NSMs by 30 vehicles scenario.

5.3.4 Proposed SCH Reservation Scheme

IEEE 1609.3 [114] and WAVE/DSRC [18] documents do not specify an algorithm for SCH assignment for different WBSS service providers (SP). Without an appropriate algorithm, multichannel operations may face inefficient channel utilization problems and hidden terminal is one of the them. Suppose two SPs are out of the transmission range of each other, thus the WSA sent from one SP cannot be received by another SP. Both SPs may reserve the same SCH in the same SCHI so that frequency-overlapping WBSSs are established. The vehicles from these two WBSSs in the overlapping zone may experience severe interference.

Cognitive radio technology is usually adopted to enable multichannel access for vehicular communications as shown in [131–133], while cognitive radio may require more complex designed PHY layer. Besides, research in [134] introduces a TDMA based SCH reservation while the authors did not provide discussions on the overhead traffic during the TDMA coordination process. Research in [135] proposed to determine the SCH with the lowest encountered packet collisions in the past. Research in [136] adaptively selects EDCA parameters depending on the density of the vehicles for improving throughput of each AC. However, none of these research works are able to reduce the hidden terminal problem. Research in [126, 137] propose the CraSCH solution in order to avoid the set up of frequency-overlapping WBSSs. With CraSCH, an *enhanced* WSA is sent by the SP containing additional information about SCH reservations. A WSA not only contains the

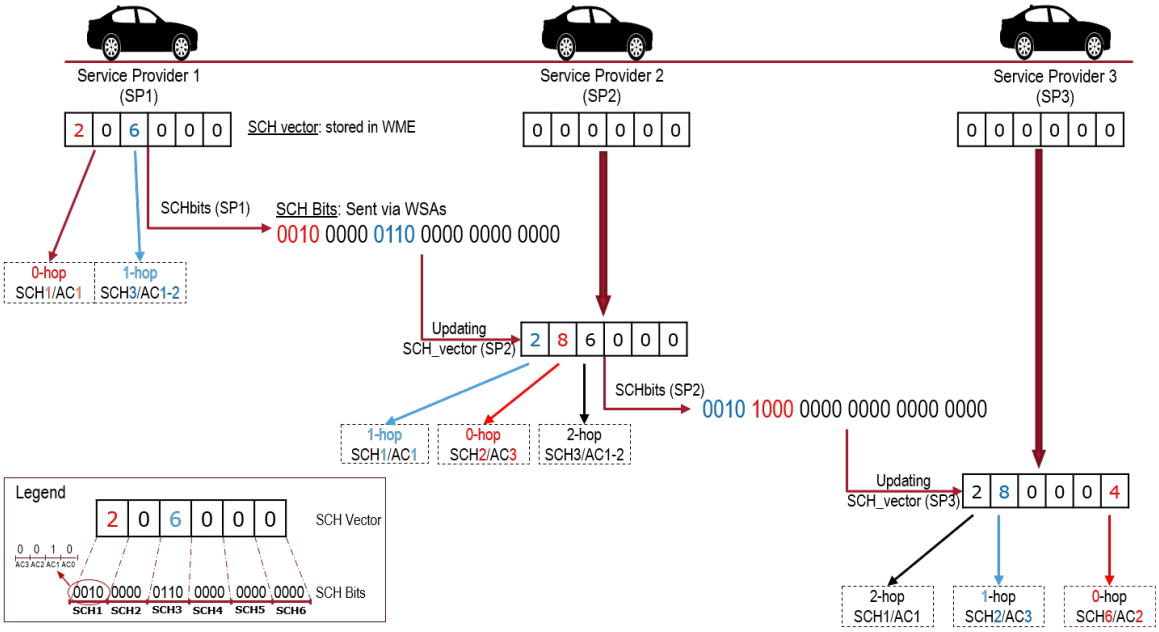


Figure 5.19: Proposed SCH/AC mix reservation scheme. Convey up to 2-hop SCH information can decrease the frequency overlapping problem. Consideration to both SCH and AC information increases the utilization of multichannel EDCA operations.

SCH number reserved by the SP, but also conveys the SCH reservation status of other WBSSs based on the WSAs received from the other SPs. Suppose SP1 selects SCH1 for its WBSS and broadcast WSA containing SCH1 information. When SP2 receives this WSA, it is aware that SCH1 is reserved and then selects SCH2. A WSA is created informing both SCH1 are SCH2 are occupied. SP3 is out of the range of SP1 but within the range of SP2. From the WSA of SP2, SP3 knows SCH1 and SCH2 are not available so that it will choose from the rest of four SCHs. In this way, even though SP3 cannot hear SP1, it can avoid picking the same SCH with SP1. CraSCH can avoid as much as possible the frequency-overlapping WBSSs by conveying SCH occupancy information among the two-hop neighboring SPs. However, CraSCH is only effective when there are free SCHs. If all SCHs have been reserved, CraSCH just randomly chooses one of the reserved SCHs. Therefore, CraSCH loses its effect when the number of WBSSs is more than the total number of SCHs.

Based on the analysis in the above section, all four ACs can be enabled simultaneously

for different non-safety services in multiple SCHs. Thus, we proposed a SCH reservation mechanism, named as *SCHAC*, which improves the multichannel utilization by fully considering the AC status in each SCH. Our proposed approach is suitable for the situation when all SCHs have been reserved. For example, suppose a SP needs to start a non-safety service with AC1 priority and it is aware that all SCHs have been reserved. If it uses CraSCH, the SP will just randomly select one SCH to use. If the AC1 priority in this SCH just happens to be occupied, both services will contend for channel access and the utilization of channel is decreased. With our approach, the SP will further look into the status of ACs in each SCH and it will find which SCH has idle AC1 or has less utilized ACs if AC1 priorities from all SCHs have been used.

Figure 5.19 shows the progress of sharing SCHs/ACs information using our proposed approach. In the figure, three SPs need to initiate WBSSs in the next SCHI, SP2 is 1-hop distance with SP1 and SP3, while SP3 is two-hop away from SP1. This makes SP3 unable to receive WSA from SP1, thus it is possible that both SP1 and SP3 reserve the same SCH. In our approach, each vehicle owns a 1×6 *SCH vector* to maintain SCH status from received WSAs. Each element indicates the status of one SCH, *i.e.,* SCH1 to SCH 6 from left to right. The decimal vector element can be converted into 4-digit binary number corresponding to 4 EDCA ACs, *i.e.,* AC3 - AC0 from left to right. For example, if AC0 and AC2 have been reserved for SCH1, the element is 0101 in binary format and 5 in decimal. In Figure 5.19, the blue 6 in SP1's SCH vector indicates AC1 and AC2 of SCH3 have been reserved. SP1 obtains this information from the WSAs received before. Then SP1 decides to establish a WBSS in SCH1 with AC1 priority. To share this information, a 3 byte (24 bit) SCHbits field is created. The SCHbits field is the binary format of SCH vector, every 4 digits corresponding to the 4 EDCA ACs in one SCH. SP1 set SCH1/2 to the SCHbits field along with SCH3/6, and broadcast the SCH information with WSA messages. Note that the chosen SCH, as well as the priority of service, is also conveyed by WSA main field; this is the original function WSA, *i.e.,* both WSA and WSA_ext SCH bits fields contain SCH/AC information selected by SP1. In this way, the WSA receivers are able to distinguish which SCH/AC information is from 1-hop distance and which is from multi-hop distance.

The initial SCH vector of SP2 is all zero, after receiving the WSA from SP1, SP2 is

aware that SCH1/2 is the service information from SP1 and SCH3/6 is from other SPs. To SP2, SCH1/2 is the 1-hop information and SCH3/6 is the 2-hop information. Since both SCH1 and SCH3 are occupied, SP2 picks SCH2 for its service with priority of AC3, *i.e.,* SCH2/8. Then, SP2 broadcasts WSA with SP2 SCH information SCH2/8 and 1-hop SCH information SCH1/2. Note that SCH3/6 is 3-hop information to other SPs, thus it is not included in SP2's WSA. Broadcasting more than 2-hop SCH information may cause wasted bandwidth and inefficient channel reuse [126]. When SP3 receives this WSA, it knows SCH1/2 is from 2-hop nodes and SCH2/8 is from SP2, then it selects SCH6/4 to set up its WBSS on SCH6 with priority AC2. In this way, SP3 is able to obtain the SCH reservation information of SP1 even though SP1 is 2-hop away.

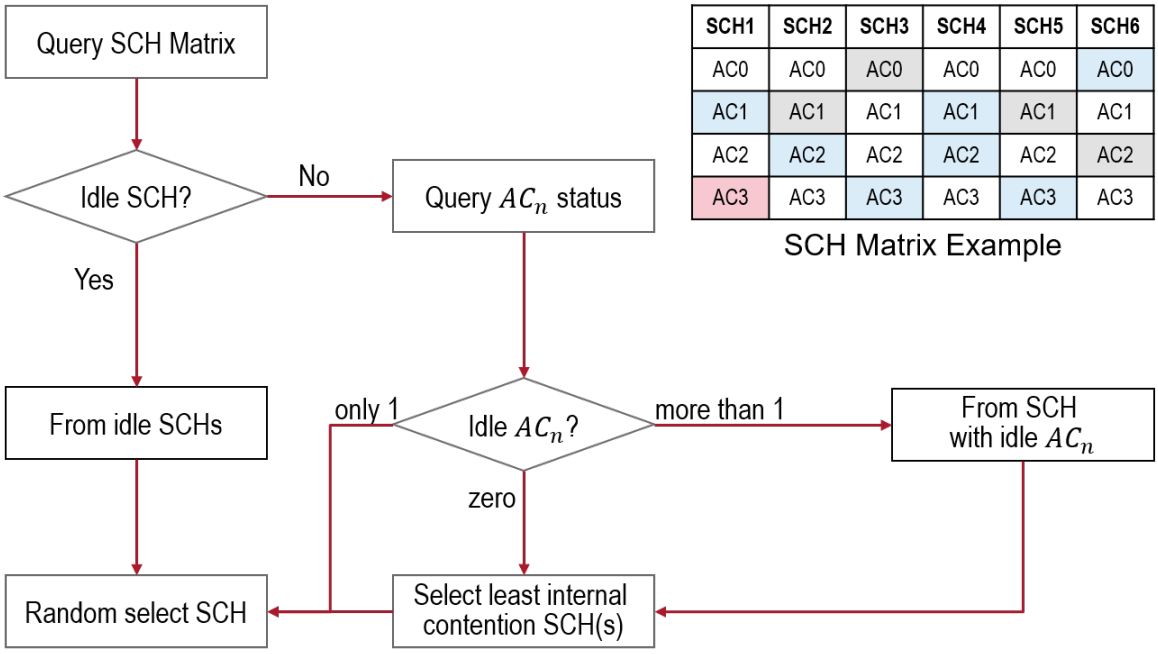


Figure 5.20: Latency improvement by the proposed SCH/AC reservation scheme. When the number of WBSSs outweighs the number of SCHs, proposed SCH/AC reservation scheme can cause a much lower latency than random pick scheme.

Figure 5.20 illustrates the process of selecting a SCH while considering the AC status of each SCHs. Suppose a SP needs to establish a WBSS with the AC3 priority and it is aware that all SCHs have been reserved by other SPs within 2-hop distance after querying the local SCH matrix. The SP will continue to check the AC status stored in the SCH matrix.

During this process, this SP may face with three choices. First, only one SCH out of six SCHs has a vacant AC3 position, the SP will select this SCH. Second, more than one SCHs have vacant AC3, the SP will choose the one with the smallest AC values amongst the SCHs who have available AC3. Suppose the SCH/AC information of the SP is the same with the SCH Matrix Example shown in Figure 5.20, the SP will select SCH1 since SCH1 only has AC1 being reserved while SCH3, SCH5 do not have available AC3 and SCH2, SCH4 and SCH6 have more than one ACs being reserved. Third, if the AC3 from all SCHs have been reserved, the SP will pick the SCH with the least AC total values since a smaller overall AC value indicates a lower or fewer ACs being reserved. Furthermore, the SCH vector is reset to zero at the beginning of every CCH interval.

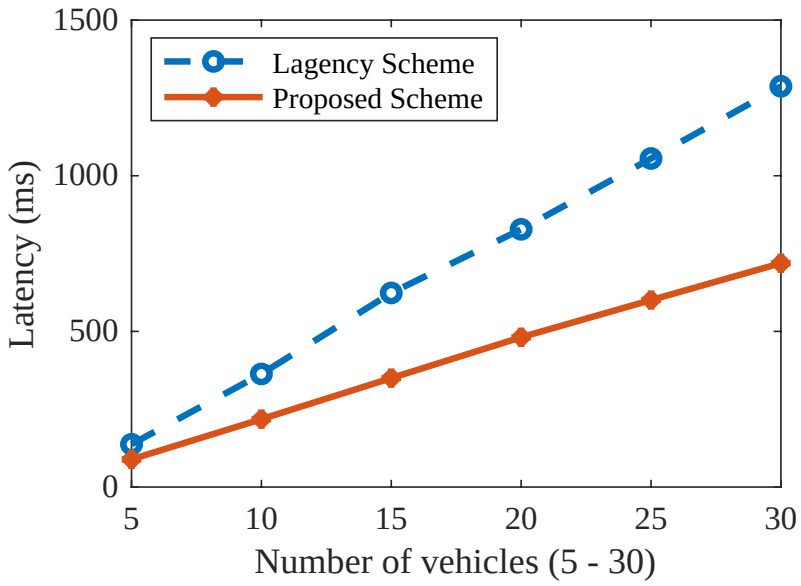


Figure 5.21: Latency improvement by the proposed SCH/AC reservation scheme. When the number of WBSSs outweighs the number of SCHs, proposed SCH/AC reservation scheme can cause a much lower latency than random pick scheme.

Simulations are performed to test the proposed scheme. As the SCH reservation depends on the establishment of WBSSs, which is affected by the total number of vehicles, we increase the number of vehicles from 5 to 30. Random vehicles are chosen to be SPs, the other vehicles have the option to choose which WBSS to join if more than one WBSSs coexist.

Once a WBSS is established, each member needs to broadcast 30 messages with one of AC priorities to finish the service. Two sets of experiments are conducted, one group is using the legacy SCH random pick scheme, another group is using the proposed SCH/AC mix scheme. In order to show the difference more intuitively, the average latency is calculated by all data from AC0 to AC3. The results are shown in Figure 5.21.

When the number of vehicles is low, the difference between two SCH reservation schemes are not obvious. This is because the number of WBSS is low due to the lack of vehicles. Even with random selection, the WBSSs have a high probability to choose different SCHs. As the number of vehicles increases, our proposed scheme shows its advantage. With 30 cars, all six SCHs are reserved with partial ACs occupied, the proposed scheme can save nearly half time of random pick schemes.

5.4 Chapter Summary

In this chapter, we implemented multichannel (MC) feature by extending the existing single channel models. The APP (MC) layer is able to initiate or join a non-safety service via a WBSS. The messages generated by APP (MC) DES module contain multichannel information. The MAC (MC) DES module is expanded by making use of multiple single channel MAC DES modules. A new DES module, WME, is designed to maintain multi-channel status and information. WME DES module is also responsible to coordinate with APP (MC) DES module for generating WSA frames. Further, the WME DES module controls the MAC (MC) module to switch on or off the sub-MAC modules depending on the channel information.

Several experiments have been conducted in order to evaluate the performance between single channel and multichannel communication. All four ACs from EDCA scheme have been enabled in the simulations, *i.e.,* the data flows within 4 ACs $\times$ 7 channels = 28 ACs, the results show that data of AC0-2 are experiencing severe long delay and are not qualified for transmitting safety messages in single channel V2V communication. Then only AC0-1 are set for non-safety applications, while AC2-AC3 are granted to safety-related applications. The results show that AC2 is still facing long delays that cannot be used on

transmitting safety messages in single channel communication. Meanwhile, due to the even lower latency, AC0-AC1 are experiencing even longer queue delay that cannot provide a high throughput for non-safety services. On the contrary, the multichannel communication is able to keep safety-messages at a relatively low latency while providing high throughput for non-safety messages. The simulations also show that for time-sensitive services, the transmitting time should be carefully scheduled in order to avoid latency caused by channel switching operation. Furthermore, the message cause by random event may have long latency caused by channel switching operation, and should be compensated by other technologies. Finally, We proposed a cooperative SCH reservation criteria, which could adaptively decide the SCH frequency as well as AC priority according to the SCH/AC information received from nodes with at most two-hop distance. The simulation result shows that the proposed scheme is able to increase the utilization of multichannel with multiple priorities, thereby reduce the contention probability to achieve an improvement on packet delivery latency.

Chapter 6

Research Achievement and Future Work

In this dissertation, we have presented the design of multichannel vehicular network simulator using discrete-event programming. We developed an accurate PHY layer at the bit level, a MAC layer DES module that supports EDCA scheme, and an APP layer DES module that integrates vehicle mobility models with the application message generation actions. Then the simulator is expanded to supported multichannel operations including channel switching and coordination operations. The DES framework introduced in this this dissertation are practical and compatible with any other discrete-event programming languages.

6.1 Research Achievements

In this dissertation, several achievements have been made in the area of both single channel and multichannel scenarios within a vehicular network. And the skills on analyzing standards and system-level design on the individual functions are also improved. The achievements are summarized as follows:

- **Bit-level processing:** The PHY layer that supports bit-level processing has been evaluated and the performance has been compared with the well-known packet-level

network simulator, *NS-3*. The simulation results show the bit-level processing is able to model a more realistic and accurate wireless channel by maneuvering each single symbol inside a waveform than packet-level simulators.

- **Performance of V2V based on DSRC:** The performance of EDCA is evaluated by simulations created by *VANET Toolbox*. The simulations show that data transmission from multiple AC queues can coexist in the same channel (CCH or SCH), additionally, the AC3 communication maintains a nearly 100% packet delivery rate (PDR) and much lower latency. When the number of vehicles are less than 15, AC2 performs better than CSMA. When the number of vehicles are more than 15, CSMA acts better than AC2 transmission. However, AC2 still has a 85% PDR with an acceptable latency. The simulations prove that AC2 is capable to transmit BSMs with acceptable packet collisions quantities and AC3 is good enough to ensure the quality of service (QoS) of emergency (EMG) messages.

- **Lane Changing Schemes:** By making use of BSMs and EMG messages, two lane changing schemes are proposed. Vehicles obtain other vehicular traffic information via BSMs. The lane changing (LC) information are exchanged LC requests and replies. All LC messages are at the AC3 priority. In addition, reliable data transmission (RDT) has evolved to guarantee the LC safety. Simulations are created using the *VANET Toolbox*, and the results prove that the proposed LC schemes can increase the overall traffic efficiency while maintaining safety.

- **Multichannel operations:** The EDCA scheme is fully enabled with four ACs in multichannel scenario. The simulations show that multichannel communication is necessary to ensure a low latency and high packet delivery ratio to safety-related services and a high throughput for non-safety services simultaneously. Safety-related services with strict latency requirements should be carefully scheduled during CCHI. Messages caused by random emergency event may experience severe delay due to channel switching operation.

- **SCH/AC Reservation Scheme:** A coordinated SCH/AC reservation scheme is

proposed in order to increase the SCH utilization. The SCH/AC information within 2-hop distance is shared via WSA frames. The simulation results show that the proposed SCH/AC scheme is able to reduce the contention probability so that to gain a higher channel utilization.

- **VANET Toolbox:** *VANET Toolbox* is the first vehicular network simulator in MATLAB/Simulink environment. Before MATLAB DES was published, MATLAB/Simulink is a time-driven environment and does not support multi-threading programming. This limitation makes it incompetent to create a synchronous network simulator. Implemented by MATALB DES, *VANET Toolbox* supports a hybrid of time-driven and event driven simulation environment. Then, *VANET Toolbox* is an integrate type simulator because the mobility models are integrated with APP layer of the vehicular network simulator. Furthermore, *VANET Toolbox* can be extended to work with other MATLAB/Simulink softwares such as 'WLAN System Toolbox' or communication hardwares such as USRP.

6.2 Future Works

The future work of this PhD research can be categorized into two categories. In the first category, we will be extending the proposed simulator to more complicated scenarios including vehicle-to-infrastructure (V2I) and vehicle-to-everything (V2x) communication. Moreover, wireless channel models such as urban non-line-of-sight(NLOS) will be developed. Based on these developments, the city crossroads with traffic lights can be simulated and more services such as left-turn assistant (LTA) can be developed and evaluated. The second category of the future works is to implement the proposed connected vehicular communication in the radio hardware, such as the USRP software defined radio or Atheros 9k Wi-Fi modules. Currently, implementing the full stack of vehicular networks into hardware is still an open research topic since the current implementations focus on the wireless communication, *i.e.,* the PHY layer. The event-based activities from the higher layer such as multichannel switching operation and coordination have yet been implemented. The proposed bit-level PHY layer with the DES-based upper layer is convenient to be converted to

hardware implementations.

Appendix A

Classifications of Systems

The Concept of System

A *system* is a set of interacting components which behave together to perform a function and this function cannot be performed by any of the individual parts [138]. A system can be abstracted into a model, indicated as $g(\cdot)$ and shown in Eq. (A.1) [53].

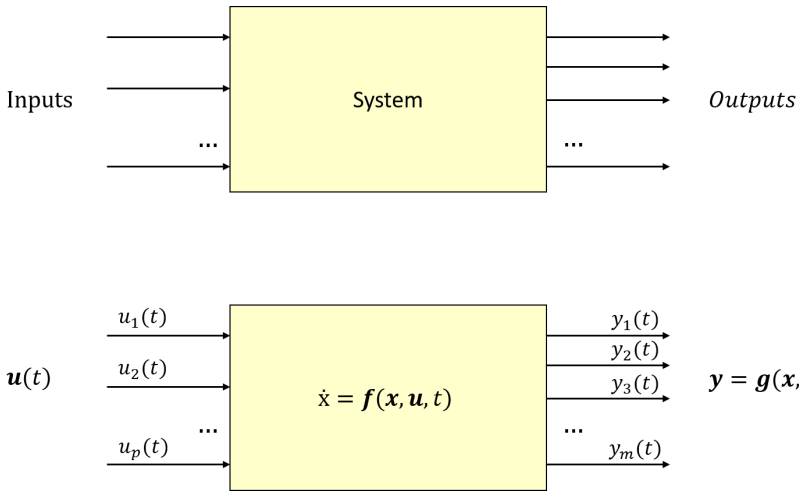


Figure A.1: A system with I/O and state space model process. The upper plot shows a system with inputs and outputs. The plot below is the abstraction of the above system. The output vectors are decided by system function, input vector, time and state space model.

$$Input : \vec{u}(t) = [u_1(t), u_2(t), ..., u_p(t)]$$

$$Output : \vec{y}(t) = [y_1(t), y_2(t), ..., y_m(t)]$$

$$\vec{y} = \vec{g}(\vec{u}) = [g_1(u_1(t), u_2(t), ..., u_p(t)), ..., g_m(u_1(t), u_2(t), ..., u_p(t))]. \quad \text{(A.1)}$$

A basic input-output model is shown in Figure A.1. In the figure, $\vec{u}(t)$ is a vector of inputs, $\vec{y}(t)$ is a vector of outputs and $\vec{x}(t)$ is a vector of states. The state space $\dot{x}$ is a function decided by state vector, input vector and time. The output $\vec{y}$ is calculated by $\vec{y} = \vec{g}(\vec{x}, \vec{u}, t)$. The classifications of systems are shown in Figure A.2

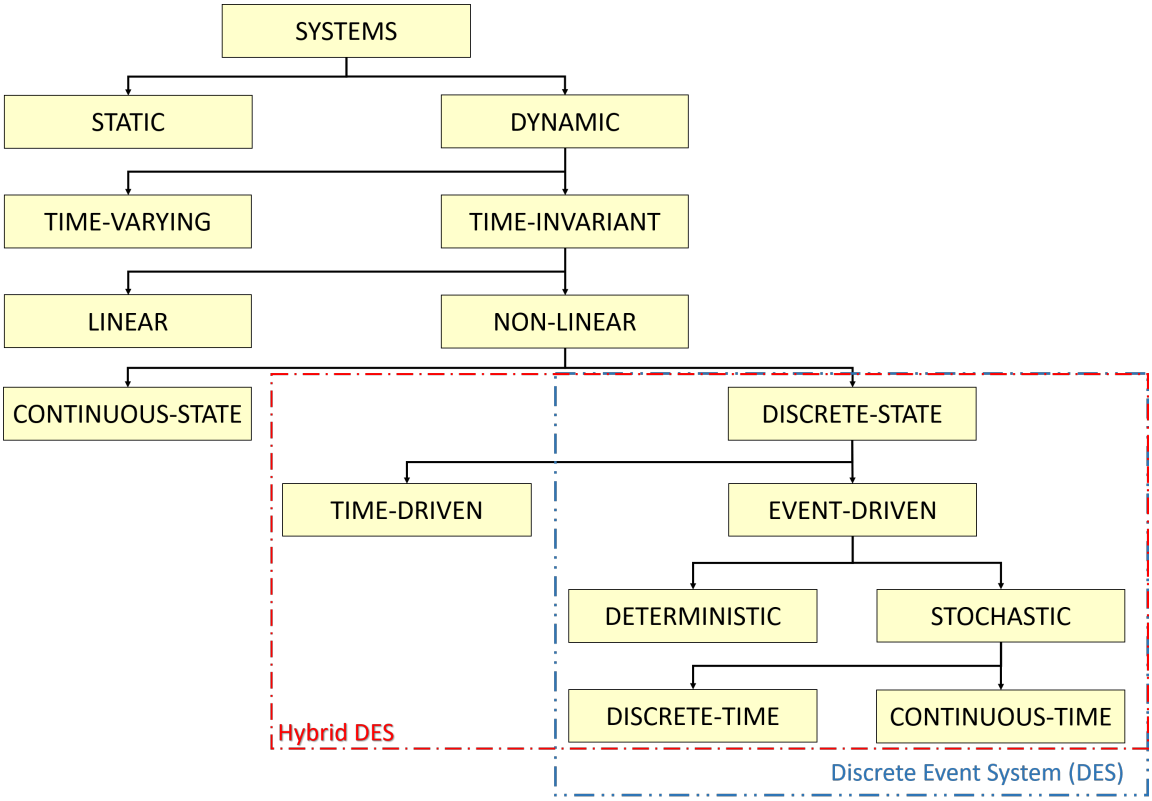


Figure A.2: Systems overview and Discrete Event Systems. Discrete Event System is classified as a discrete-state event-trigger system. A hybrid system including both time-driven and event-driven DES is suitable for network PHY layer simulation.

Static and Dynamic System

A *static* system is defined when the output $\vec{y}(t)$ is independent of past values of the input $\vec{u}(\tau), \tau < t$ for all t. When the output of a system depends on the past values of the input, *i.e.,* the output requires buffer of the input history, this system is *dynamic*. Differential

equations, for continuous-time system, or difference equations, for discrete-time systems, are usually chosen to model the behavior of dynamic systems.

Time-Varying and Time-Invariant System

When the system $\vec{g}(\cdot)$ is not independent of time t, Eq. (A.1) is represented into Eq. (A.2).

$$\vec{y} = \vec{g}(\vec{u}, t). \tag{A.2}$$

In a *time-invariant system*, if $\vec{y}(t) = \vec{g}(\vec{u}(t))$, then $\vec{y}(t-\tau) = \vec{g}(\vec{u}(t-\tau))$, *i.e.,* if the input is τ time later than t, the resulting output should be the same result as obtained at t, otherwise the system is *time-varying system*. The behaviors of a time-invariant system do not change with time, whenever a specific input is applied to the same time-invariant system, the response is always in the same way.

Linear and Nonlinear System

For a system represented by $\vec{y} = \vec{g}(\vec{u})$, the function g is *linear* when satisfying $g(a_1u_1 + a_2u_2) = a_1g(u_1) + a_2g(u_2)$. Similarly, Eq. (A.3) shows the vector case of linear system.

$$\vec{g}(a_1\vec{u_1} + a_2\vec{u_2}) = a_1\vec{g}(\vec{u_1}) + a_2\vec{g}(\vec{u_2}). \tag{A.3}$$

A linear time-invariant (LTI) dynamic system is described by the state space model of Eq. (A.8) and Eq. (A.9).

State

At a specific time, a system's behavior can be described in a measurable way, i.e., state. The state of a system at time t_0 is defined as the output y(t) of a system for all $t >= t_0$ is uniquely determined by the system status at t_0 and system input $\vec{u}(t)$, $t >= t_0$. The state variables are defined as $\vec{x}(t) = [x_1(t), ..., x_n(t)]$.

Given the initial condition $\vec{x}(t_0) = x_0$ and the input $\vec{u}(t)$ for all $t >= t_0$, the state $\vec{x}(t)$ is presented by *state equations* shown in Eq. (A.4). Then the output $\vec{y}(t)$ is determined by

state equations, *input* and *time* shown in Eq. (A.5). The input-output model with state space is shown in Figure A.1.

$$\dot{\vec{x}}(t) = f(\vec{x}(t), \vec{u}(t), t). \tag{A.4}$$

$$\vec{y}(t) = \vec{g}(\vec{x}(t), \vec{u}(t), t). \tag{A.5}$$

For a static system, the state remains fixed at all time, *i.e.*, $\dot{x}(t) = 0$ for all t, then the system model is specified only by Eq. (A.5). For a time-invariant system, neither function $\vec{f}(\cdot)$ nor $\vec{g}(\cdot)$ explicitly depends on time t, the state equations can be written to $\dot{\vec{x}}(t) = f(\vec{x}(t), \vec{u}(t))$ and the output $\vec{y}(t) = \vec{g}(\vec{x}(t), \vec{u}(t))$. Considering linearity, Eq. (A.4) and Eq. (A.5) are presented as Eq. (A.6) and Eq. (A.7) respectively,

$$\dot{\vec{x}}(t) = \vec{A}(t)\vec{x}(t) + \vec{B}(t)\vec{u}(t). \tag{A.6}$$

$$\vec{y}(t) = \vec{C}(t)\vec{x}(t) + \vec{D}(t)\vec{u}(t). \tag{A.7}$$

in which the dimension of $\vec{A}(t)$ is $n \times n$, $\vec{B}(t)$ is $n \times p$, $\vec{C}(t)$ is $m \times n$ and $\vec{D}(t)$ is $m \times p$. In a time-invariant system, *time* element is constant, therefore Eq. (A.6) and Eq. (A.7) are simplified to Eq. (A.8) and Eq. (A.9).

$$\dot{\vec{x}} = \vec{A}\vec{x} + \vec{B}\vec{u}. \tag{A.8}$$

$$\vec{y} = \vec{C}\vec{x} + \vec{D}\vec{u}. \tag{A.9}$$

Continuous-State and Discrete-State Systems

The state space of a system, denoted by X, is a set of all possible values a state may take. Based on the type of states in a model, a system can be classified into continuous-state system and discrete-state system. In a continuous system, the state space X are continuous and can take on any, real or complex, value. In a discrete-state system, the states are all non-negative integers of a discrete set and only allowed to change from one discrete state value to another.

The analysis of a continuous-state model can be ultimately reduced to the analysis of differential (continuous-time system) or difference (discrete-time system) equations. But the mathematical analysis to express and solve a discrete-state system is relatively complicated.

Time-Driven and Event-Driven Systems

In a continuous-state system, the states change as the time changes. The time variable, t in continuous time or k in discrete time, enable the system to transit from one state to another state continuously. The system with such property is referred as *time-driven system*. A continuous-state system is by nature time-driven.

While in a discrete-state system, the state transitions are either synchronized by a clock or occurred asynchronously at some special time point due to instantaneous state transitions. These state transitions are associated with *events*. An event, denoted by e, occurs instantaneously and cause transitions from one sate value to another. A *discrete event set* E is defined as a discrete set with all these events as elements.

If state transitions in a discrete-state system are synchronized by clock ticks, it is a discrete-state, time-driven system. Otherwise, if state transitions occur asynchronously at various random time instants due to the combination of event processes, this system is referred as a discrete-state, event-driven system.

Deterministic and Stochastic Systems

A *deterministic* system is defined when no output is random. In a deterministic system, given an input $\vec{u}(t)$ for all $t >= t_0$, the state $x(t)$ can be evaluated. Whenever one or more system outputs is a random variable, the system is a *stochastic* system. In a stochastic system, the overall state is a random process, *i.e.,* the state at time t is aa random vector. The probability distribution function (PDF) of the state is required to model the system behavior.

Continuous-Time and Discrete-Time Systems

In a continuous-time system, *time* is a continuous variable. A continuous-time model can be analyzed by differential equations such as Eq. (A.4) and Eq. (A.5). In contrast

to the continuous-time systems, the time line in a discrete-time model is a sequence of intervals. A discrete-time system does not imply the discretization of the state space X, *i.e.,* the states may be continuous in a discrete-time system.

In a discrete-time system, the *time* element t is replaced by k. Similarly, the input $\vec{u}(t)$ and output $\vec{y}(t)$ are replaced by $\vec{u}(k)$ and $\vec{y}(k)$. The state variables $\vec{x}(t)$ is replaced by $\vec{x}(k)$. Furthermore, the differential equations (A.4) - (A.5) used in the continuous-time system become the following difference equations.

$$\vec{x}(k+1) = f(\vec{x}(k), \vec{u}(k), k). \tag{A.10}$$

$$\vec{y}(k) = \vec{g}(\vec{x}(k), \vec{u}(k), k). \tag{A.11}$$

For linear discrete-time systems, Eq. (A.7) and Eq. (A.6) are replaced by Eq. (A.12) and Eq. (A.13).

$$\vec{x}(k+1) = \vec{A}(k)\vec{x}(k) + \vec{B}(k)\vec{u}(k). \tag{A.12}$$

$$\vec{y}(k) = \vec{C}(k)\vec{x}(k) + \vec{D}(k)\vec{u}(k). \tag{A.13}$$

which are simpled in the time-invariant discrete-time system to Eq. (A.14) and Eq. (A.15).

$$\vec{x}(k+1) = \vec{A}\vec{x}(k) + \vec{B}\vec{u}(k). \tag{A.14}$$

$$\vec{y}(k) = \vec{C}\vec{x}(k) + \vec{D}\vec{u}(k). \tag{A.15}$$

Discrete Event Systems

When a system can be described as a set of discrete states, and the state transitions are caused by events which occur instantaneously, this system is a Discrete Event System (DES), *i.e.,* a DES is a *discrete-state*, *event-driven* system, and its state transition is entirely caused by the asynchronous discrete events over time as shown in Figure A.2. In a DES, 'time' is no longer the key factor to drive the system. The set of *events* replace the role of *time* and serves the purpose of driving a DES, each 'event' can cause a state transition. A DES may be modeled in either continuous time or in discrete time.

Hybrid Discrete Event System

Even though a DES is defined as a discrete-state, event-driven dynamic system, it is worth noting that a hybrid DES is more general when both time-driven and event-driven are present, as shown in Figure A.2. For example, the operating system (OS) in a computer is designed to not only respond to asynchronous events occurred at any time but also process functions synchronized by the computer clock. A hybrid DES may be deterministic or stochastic and it can be modeled in either discrete or continuous time.

Appendix B

An Example of MATLAB DES Model

In this section, we create a simplified MAC layer to show how to create a MATLAB discrete-event system (DES) with the necessary methods mentioned in the above sections. The sample MAC layer is highly simplified and only have partial functions. The purpose of this example is to show the skeleton of a basic MATLAB DES. This basic MATLAB DES is necessary for readers to understand the design of *VANET Toolbox* which will show in Chapter 3.

Description of a Simple DES Model

The DES model contains an *entity generation*, an *entity terminator* and a MATLAB DES. The entity generator acts as an APP layer and generates *payload* entities every 4 seconds. The payloads are sent to the MATLAB DES, in which they are converted into frames. The frames are sent to the *entity terminator*, which destroys all received frames.

The MATLAB DES works as the MAC layer. It receives payloads from the input port, and generates frames based on the information from the message part of payload. The conversion between payload and frame costs a time delay period. After the delay, the frame is generated and forwarded to the output port. The abstraction of the simplified MAC layer is shown in Figure B.1.

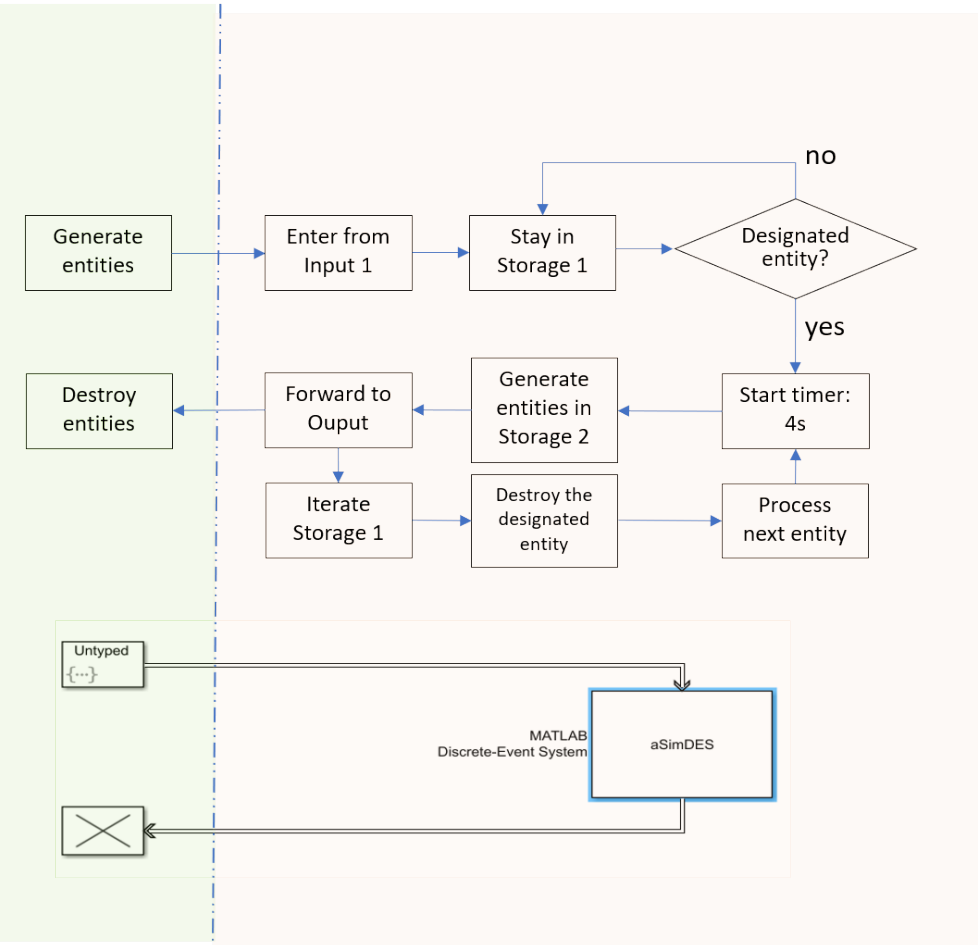


Figure B.1: The flowchart of entities inside the DES storages. The upper section shows the data flow happened inside of the below MATLAB DES.

In the figure, the structure of flowchart corresponds to the whole DES model in Simulink. As the name suggested, entity generator is for generating entities and entity terminator is to destroy entities. The MATLAB DES block is a user-defined DES system object contained in a Simulink *system block* so as to work with other Simulink blocks.

The MAC DES has two storages, one is for payload type of entities and another is for frame entities. Payloads enters from *input port 1* and stays in *storage 1*. If the sequence number in the payload header equals to the *Target payload* property, starts from 1, this payload is the designated payload. Thus a timer is attached to this payload entity. Once the delay period of the timer is ended, a frame containing the designated payload is generated

in *storage 2*. Then this frame is forwarded to the output port 1, meanwhile an iteration is triggered in storage 1 upon the frame exits storage 2. The iteration first finds the old designated payload and destroy it. Then the new designated payload counter increases by 1 and find the next designated payload in storage 1. When the new designated payload is found, the iteration process stops and the MAC DES repeats the framing process to the new payload, so on and so forth.

Create Bus Data Type

In this example, the MAC DES model is using *bus data type*. Two types of buses, Payload and Frame, are defined in the MATLAB code. Payload bus has two elements, *message* and *payloadHeader*. Payload.payloadHeader contains sequence number in a 1×1 real data field. And Payload.message saves a $[1 \times 100]$ numerical message. Similarly, Frame bus owns two fields, *Frame.frameHeader* saves the frame sequence number and *Frame.body* contains the message extracted from payloads.

The script, named as *init_aSimDES*, needs to be initialized before the DES simulation starts. Users can run init_aSimDES manually everytime before running the Simulink model. A more efficient way is putting init_aSimDES in the *InitFcn* of Simulink callbacks, thus init_aSimDES will be running automatically during the initialization phase of a Simulink model. The code is shown below:

```
Payload=Simulink.Bus;
payload1=Simulink.BusElement;
payload1.Name='message';
payload1.Dimensions=[1,100];
payload2=Simulink.BusElement;
payload2.Name='payloadHeader';
payload2.Dimensions=[1,1];
Payload.Elements=[payload1,payload2];

Frame=Simulink.Bus;
frame1=Simulink.BusElement;
frame1.Name='frameHeader';
frame2=Simulink.BusElement;
frame2.Name='body';
frame2.Dimensions=[1 100];
Frame.Elements=[frame1,frame2];
```

Entity Generator Block

Entity Generator generates entities at a constant or variable rate despond on the configuration. As shown in Figure B.2 (left), the entity generator generates entities every 4 seconds. The second tab of entity generator controls the entity type, in this example, 'Bus object' is selected, shown in Figure B.2 (right). Thanks to init_aSimDES, two bus object, 'Payload' and 'Frame', have been created. In the field of *Entity type name*, 'Payload' is chosen, so that the entity generator is able to generate 'Payload' type entities.

In the entity generator block, users can setup some initial values to the entity in the *Event actions* field. In this example, we give an index to the sequence number in Payload.payloadHeader field. The value, started from 1, increases by 1 as more entities are generated. The code is shown below:

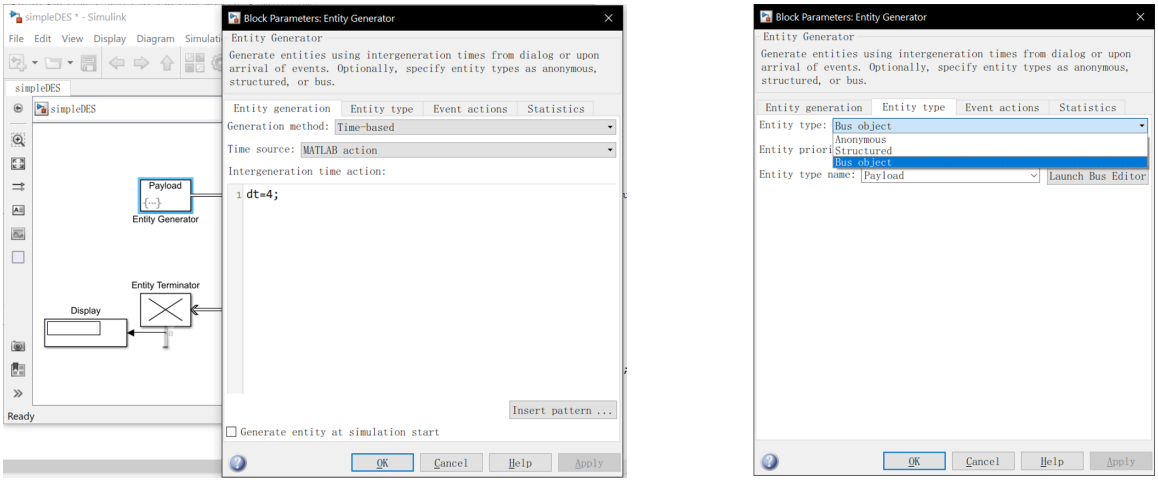


Figure B.2: Entity Generator Block from SimEvents Library in Simulink and Configuration. The entity intergeneration rate is set to a constant rate and the entity type it creates are all 'bus' type.

```
persistent i
if isempty(i)
i=1;
end
entity.payloadHeader=i;
i=i+1;
```

MATLAB DES block

MATLAB Discrete-Event System block is the main block for the whole model, shown in Figure B.3. As described in the above tutorial sections, we need to first define the data type of input/output port as well as the storages using the code below:

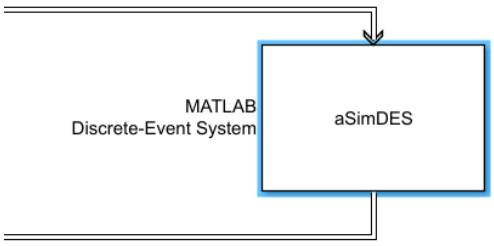


Figure B.3: The Discrete Event System (DES) Block in Simulink.

```
function entityTypes=getEntityTypesImpl(obj)
entityTypes=[obj.entityType('payload','Payload'),...
obj.entityType('frame','Frame')];
end
function [inputTypes,outputTypes]=getEntityPortsImpl(~)
inputTypes={'payload'};
outputTypes={'frame'};
end
function [storageSpecs,I,O]=getEntityStorageImpl(obj)
storageSpecs=[obj.queueFIFO('payload',inf)...
obj.queueFIFO('frame',inf)];
I=1;
O=2;
end
```

In the code, two entity types, 'payload' and 'frame', are extended from the corresponding bus type 'Payload' and 'Frame'. The DES has one input port in 'payload' type and one output port in 'frame' type. Two storages are defined and both of them are a FIFO queue with infinite capacity. Storage 1 is 'payload' type to contain payloads and storage 2 is

'frame' type to store payload entities. Storage 1 is connected with input port, thus the payload entities enter the DES via input port and are saved in storage 1 directly. Storage 2 is connected with output port, for all entities left storage 2, they leave the DES block too.

When a payload entity enters storage 1, it triggers *action payloadEntryImpl( )*. Due to the requirement of code generation, *payloadEntryImpl()* is renamed to *payloadEntry()*. All the other action methods follow the same rule too. One thing needs to mention is that the example of this DES is highly simplified, thus not all input parameters are used. we try to use *disp* to illustrate the functions of most parameters. The code is shown below:

```
function [entity,events] = payloadEntry(obj,storage,entity,source)
disp(['T=' num2str(obj.getCurrentTime()) 's: payload ' num2str(
    ↪ entity.data.payloadHeader) ' enters storage ' num2str(storage
    ↪ ) ' from ' source.type ' ' num2str(source.index)]);
if entity.data.payloadHeader == obj.TargetPayload
events=obj.eventTimer('framing',5);
end
end
```

In the above code, once an entity enters the storage, the DES compares its sequence number (payloadHeader) with the predefined property. If the entity is the designated entity, it triggers the timer event *eventTimer* with tag of *'framing'*. Otherwise, it keeps waiting in storage 1 until further called. The timer event delays 5 seconds to the designated entity. Once the delay period is ended, the corresponding action *payloadTimer* is triggered. The code of *payloadTimer* action is shown below:

```
function [entity,events]=payloadTimer(obj,storage,entity,tag)
if strcmp(tag,'framing')
disp(['T=' num2str(obj.getCurrentTime()) 's: frame ' num2str(entity.
    ↪ data.payloadHeader) ' is generated.']);
obj.payloadHeaderBuffer=entity.data.payloadHeader;
obj.frameBodyBuffer=entity.data.message;
events=obj.eventGenerate(2,'generateFrame',0,10);
end
end
```

Please note that *strcmp* is not necessary to the functionality DES, the reason it is used in the code is that we want to show the feature of 'tag'. In a more complicated DES, the tag can be compared by *strcmp* or *switch ... case...end*. In the payload timer action, the payload Header and message are buffered to the properties of DES for further. Then a frame generate event is called targeting storage 2 with tag of 'generateFrame'. This event create a frame entity inside storage 2 with 0 delay. A frame generate action is triggered upon the creation of the frame. The code is showing below:

```
function [entity,events]=frameGenerate(obj,storage,entity,tag)
if strcmp(tag,'generateFrame')
entity.data.body=obj.frameBodyBuffer;
entity.data.frameHeader=obj.payloadHeaderBuffer;
events=obj.eventForward('output',1,0);
end
end
```

In the *frameGenerate* action, the pre-saved sequence number and message extracted from payload are given to the frameHeader field and body field correspondingly. Then the frame is sent to the output port via *eventForward()* event. Before the entity left the DES, *frameExit( )* action is called, the code is shown below:

```
function [events]=frameExit(obj,storage,entity,dst)
disp(['T=' num2str(obj.getCurrentTime()) 's: frame ' num2str(entity.
    ↪ data.frameHeader) ' is forwarded from storage ' num2str(
    ↪ storage) ' to ' dst.type ' ' num2str(dst.index) '.'] );
if storage==2 && dst.index==1
events=obj.eventIterate(1,'findNextPayload',1);
end
end
```

Similar to *frameGenerate* code, the comparisons (storage ==2 && dst.index==1) are unnecessary to the functionality of the DES except to show the use of 'storage' and 'dst'. While this entity is forwarding to the output port, an iteration is triggered by *eventIterate()* event with tag of 'findNextPayload'. The iteration is in storage 1, thus it is payload type. The corresponding payload iterate action is shown below:

```
function [entity,events,next]=payloadIterate(obj,storage,entity,tag,
    ↪ cur)
disp(['T=' num2str(obj.getCurrentTime()) 's: iterating on payload '
    ↪ num2str(entity.data.payloadHeader) ' in storage ' num2str(
    ↪ storage) ' with size of ' num2str(cur.size) ' on position '
    ↪ num2str(cur.position)]);
if entity.data.payloadHeader==obj.TargetPayload
events=obj.eventDestroy();
next=true;
else
events=obj.eventTimer('framing',1);
obj.TargetPayload=obj.TargetPayload+1;
next=false;
end
end
```

The iteration process first find the used designated payload and destroy it. Then iteration continues (*next=true*) until the second available payload is find in the storage as the new designated payload. This payload triggers payload timer event and start the next series of events and actions as its predecessor. Before the old designated payload is destroyed, the corresponding *payloadDestroy()* action is invoked. The destroy action does nothing except showing a message via *disp()*.

```
function events=payloadDestroy(obj,storage,entity)
disp(['T=' num2str(obj.getCurrentTime()) 's: payload ' num2str(
    ↪ entity.data.payloadHeader) ' has been destroyed. ']);
end
```

Entity Terminator block

The Entity Terminator destroys all the entities it received. An Entity Terminator is not essential but usually adopted to terminate the useless output of the DES block. In this example, the entity terminator can be removed with affecting the results except a warning may be given. Connecting a *display block*, an entity terminator can show the statistics on the number of entities it has destroyed, see Figure B.4.

Simulation Results

The text results are all generated by *disp()* function, as shown in Figure B.4. At the 4th second, the first payload enters storage 1 and is delayed 5 seconds by *payloadTimer* event. 4 seconds later, the second payload enters storage 1. As the first payload is still under processing (in the 5-second delay period), payload 2 has to wait in storage 2. At time of $9s$, the delay period is over, frame 1 is created in storage 2 based on payload 1. Immediately, frame 2 is sent out via output port 1 and iteration in storage 1 is undergoing. Payload 1 is destroyed, payload 2 takes its position and starts a new set of events and actions. Due to the limit of simulation time (10s), only frame 1 is created and sent out, payload 2 is still in the delay period. The display connected to the entity terminator also proves that only one frame reached the entity terminator. Even though the example is quite basic, it covers all events and most of common actions. The vehicular network simulator *VANET toolbox* is also created by these basic events and actions.

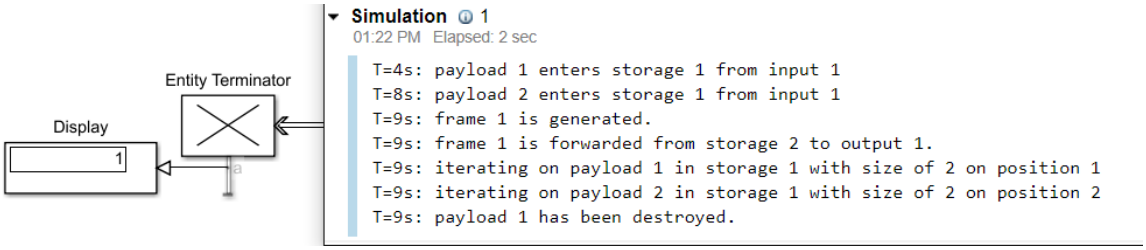


Figure B.4: The output results from the simple discrete event system. The display attached to the entity terminator shows 4 entities have been destroyed.

Appendix C

Two-Ray Gound Reflection Model

```
dRefl = sqrt(dist.^2 + (txHeight+rxHeight).^2);
sinTheta = (txHeight+rxHeight)./dRefl;
cosTheta = dist./dRefl;
reflCoeff = (-er.*sinTheta+sqrt(er-cosTheta.^2))./(er.*sinTheta+sqrt(er
    ↪ -cosTheta.^2));
Pt = 10.^(Pt./10)./1000;
d0 = 1;
Gt = 10^(Gt/10);
Gr = 10^(Gr/10);
Pd0 = Pt*Gt/(4*pi*d0^2);
E0 = sqrt(Pd0*120*pi);
d1 = sqrt((txHeight-rxHeight).^2+dist.^2);
d2 = sqrt((txHeight+rxHeight).^2+dist.^2);
c = 299792458;
freq=c/lambda;
freqAng = 2*pi*freq;
Etot = E0*d0./d1.*cos(freqAng.*(d1./c-d1./c)) + reflCoeff.*E0*d0./d2.*
    ↪ cos(freqAng.*(d1./c-d2./c));
```

```
Prec = Etot.^2.*Gr*lambda^2/(480*pi^2);
PrEfield = 10*log10(Prec)+30;
```

Bibliography

[1] Quick facts 2016. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812451

[2] L. Reger, "Securely connected vehicles - what it takes to make self-driving cars a reality," in *2016 21th IEEE European Test Symposium (ETS)*, May 2016, pp. 1–1.

[3] N. S. Yeshodara, N. S. Nagojappa, and N. Kishore, "Cloud based self driving cars," in *Cloud Computing in Emerging Markets (CCEM), 2014 IEEE International Conference on*, Oct 2014, pp. 1–7.

[4] S. Dominguez, B. Khomutenko, G. Garcia, and P. Martinet, "An optimization technique for positioning multiple maps for self-driving car's autonomous navigation," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Sept 2015, pp. 2694–2699.

[5] Waymo: google self-driving car. [Online]. Available: https://waymo.com/

[6] Ford self-driving delivery vehicle. [Online]. Available: https://www.engadget.com/2018/04/15/ford-self-driving-car-network-at-scale-in-2021/

[7] What is lidar? [Online]. Available: https://oceanservice.noaa.gov/facts/lidar.html

[8] justscience. (2017, 12) How will self driving cars change the way people live? [Online]. Available: http://www.justscience.in/articles/impact-colonoscopy-testing-colon-cancer-rates/2017/12/15

[9] C. Fernández, D. Llorca, M. Sotelo, I. Daza, A. Hellín, and S. Álvarez, "Real-time vision-based blind spot warning system: Experiments with motorcycles in daytime/nighttime conditions," *International Journal of Automotive Technology*, vol. 14, no. 1, pp. 113–122, 2013.

[10] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *ACM SIGPLAN Notices*, vol. 53, no. 2. ACM, 2018, pp. 751–766.

[11] J. Park, J. Kim, S. Kuk, Y. Park, and H. Kim, "Exploring smartphones as wave devices," in *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*. IEEE, 2015, pp. 1–5.

[12] ITS research fact sheets - benefits of intelligent transportation systems. [Online]. Available: https://www.its.dot.gov/factsheets/benefits_factsheet.htm

[13] H. Hartenstein and L. P. Laberteaux, "A tutorial survey on vehicular ad hoc networks," *IEEE Communications Magazine*, vol. 46, no. 6, pp. 164–171, June 2008.

[14] O. Tonguz, N. Wisitpongphan, F. Bai, P. Mudalige, and V. Sadekar, "Broadcasting in VANET," *2007 Mobile Networking for Vehicular Environments, MOVE*, no. June, pp. 7–12, 2007.

[15] N. Akhtar, S. C. Ergen, and O. Ozkasap, "Vehicle mobility and communication channel models for realistic and efficient highway VANET simulation," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 1, pp. 248–262, Jan 2015.

[16] N. Akhtar, O. Ozkasap, and S. C. Ergen, "VANET topology characteristics under realistic mobility and channel models," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2013, pp. 1774–1779.

[17] *IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Multi-channel Operation Corrigendum 1: Miscellaneous Corrections*, IEEE Std., Dec 2014.

[18] M. Muller, "WLAN 802.11p measurements for vehicle to vehicle (V2V) DSRC," *Application Note Rohde & Schwarz*, vol. 1, pp. 1–25, 2009.

[19] J. B. Kenney, "Dedicated short-range communications (DSRC) standards in the united states," *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1162–1182, 2011.

[20] R. A. Uzcátegui, A. J. De Sucre, and G. Acosta-Marum, "WAVE: A tutorial," *IEEE Communications magazine*, vol. 47, no. 5, 2009.

[21] Vehicle-to-vehicle/vehicle-to-infrastructure control. [Online]. Available: http://ieeecss.org/sites/ieeecss.org/files/documents/IoCT-Part4-13VehicleToVehicle-HR.pdf

[22] J. Gozálvez, M. Sepulcre, and R. Bauza, "Ieee 802.11 p vehicle to infrastructure communications in urban environments," *IEEE Communications Magazine*, vol. 50, no. 5, pp. 176–183, 2012.

[23] *Vehicle-to-Vehicle Communications: Readiness of V2V Technology for Application*, U.S DOT Std.

[24] M. Chowdhury, "15th intelligent transportation systems world congress," *Journal of Intelligent Transportation Systems*, vol. 14, no. 2, pp. 51–53, 2010. [Online]. Available: https://doi.org/10.1080/15472451003738194

[25] F. Ahmed-Zaid, H. Krishnan, M. Maile, L. Caminiti, S. Bai, and S. VanSickle, "Vehicle safety communications - applications: System design amp; objective testing results," vol. 4, pp. 417–434, 06 2011.

[26] G. Howe, G. Xu, D. Hoover, D. Elsasser, and F. Barickman, "Commercial connected vehicle test procedure development and test results–emergency electronic brake light," Tech. Rep., 2016.

[27] G. Howe, G. Xu, D. Hoover, and D. Elsasser, "Commercial connected vehicle test procedure development and test results–blind spot warning/lane change warning," Tech. Rep., 2016.

[28] G. Howe, G. Xu, D. Hoover, D. Elsasser, and F. Barickman, "Commercial connected-vehicle test procedure development and test results–intersection movement assist," Tech. Rep., 2016.

[29] J. Harri, F. Filali, and C. Bonnet, "Mobility models for vehicular ad hoc networks: a survey and taxonomy," *IEEE Communications Surveys Tutorials*, vol. 11, no. 4, pp. 19–41, Fourth 2009.

[30] Mcity: leading the transformation to connected and automated vehicles. [Online]. Available: https://mcity.umich.edu/

[31] (2017, May) New way to test self-driving cars could cut 99.9% of validation costs. [Online]. Available: https://mcity.umich.edu/

[32] H. Hartenstein and K. Laberteaux, *VANET: vehicular applications and inter-networking technologies*. John Wiley & Sons, 2009, vol. 1.

[33] D. O. Lazaro, E. Robert, L. Lan, J. Gozalvez, S. Turksma, F. Filali, F. Cartolano, M. A. Urrutia and Krajzewicz, "An Integrated Wireless and Traffic Platform for Real-Time Road Traffic Management Solutions," *COMeSafety Newsletter*, 2010.

[34] "Simulation of urban mobility (SUMO)," http://sumo.dlr.de/wiki/Simulation_of_Urban_MObility_-_Wiki.

[35] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, December 2012.

[36] A. J. Ghandour, M. Di Felice, L. Bononi, and H. Artail, "Modeling and simulation of WAVE 1609.4-based multi-channel vehicular ad hoc networks," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012, pp. 148–156.

[37] N. Torabi and B. S. Ghahfarokhi, "Implementation of the ieee 802.11p/1609.4 DSRC/WAVE in NS-2," in *Computer and Knowledge Engineering (ICCKE), 2014 4th International eConference on*, Oct 2014, pp. 519–524.

[38] Nsnam, "NS-3 project - introduction to NS-3," https://www.nsnam.org/docs/tutorial/html/introduction.html, Feb. 2017.

[39] J. Bu, G. Tan, N. Ding, M. Liu, and C. Son, "Implementation and evaluation of WAVE 1609.4/802.11p in NS-3," in *Proceedings of the 2014 Workshop on NS-3*, ser. WNS3 '14. New York, NY, USA: ACM, 2014, pp. 1:1–1:8. [Online]. Available: http://doi.acm.org/10.1145/2630777.2630778

[40] G. Pongor, "Omnet: Objective modular network testbed," in *Proceedings of the International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*. Society for Computer Simulation International, 1993, pp. 323–326.

[41] "Introduction to traffic control interface (TraCI)," http://sumo.dlr.de/wiki/TraCI, 2017.

[42] A. Acosta, "Traci4matlab," Oct. 2016. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/44805-traci4matlab

[43] A. Gil, "Traci4matlab: User's manual," *Universidad Nacional De Colombia*, 2014.

[44] L. Bononi, M. Di Felice, G. D'Angelo, M. Bracuto, and L. Donatiello, "Moves: A framework for parallel and distributed simulation of wireless vehicular ad hoc networks," *Computer Networks*, vol. 52, no. 1, pp. 155–179, 2008.

[45] S.-Y. Wang, C. Chou, C. Huang, C. Hwang, Z. Yang, C. Chiou, and C. Lin, "The design and implementation of the nctuns 1.0 network simulator," *Computer networks*, vol. 42, no. 2, pp. 175–197, 2003.

[46] M. Killat, F. Schmidt-Eisenlohr, H. Hartenstein, C. Rössel, P. Vortisch, S. Assenmacher, and F. Busch, "Enabling efficient and accurate large-scale simulations of vanets for vehicular traffic management," in *Proceedings of the fourth ACM international workshop on Vehicular ad hoc networks*. ACM, 2007, pp. 29–38.

[47] C. Gorgorin, V. Gradinescu, R. Diaconescu, V. Cristea, and L. Ifode, "An integrated vehicular and network simulator for vehicular ad-hoc networks," in *Proceedings of the 20th European Simulation and Modelling Conference*, vol. 59, 2006.

[48] Mathworks, "802.11n packet error rate simulation for 2x2 tgn channel," May 2017. [Online]. Available: https://www.mathworks.com/help/wlan/examples/802-11n-packet-error-rate-simulation-for-2x2-tgn-channel.html

[49] G. Pei and T. R. Henderson, "Validation of ofdm error rate model in ns-3," *Boeing Research Technology*, pp. 1–15, 2010.

[50] "Ettus research - announcing the USRP e313," https://www.ettus.com/product/details/USRP-E313.

[51] Mathworks, "Packetized modem with data link layer," May 2017. [Online]. Available: https://www.mathworks.com/help/comm/examples/packetized-modem-with-data-link-layer.html

[52] Vanet toolbox: A vehicular network simulator based on des. [Online]. Available: https://github.com/lewangwpi/vanet_toolbox

[53] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[54] R. Schubert, K. Schulze, and G. Wanielik, "Situation assessment for automatic lane-change maneuvers," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 607–616, 2010.

[55] C. D. Pegden, R. E. Shannon, R. P. Sadowski *et al.*, *Introduction to simulation using SIMAN*. McGraw-Hill New York, 1995, vol. 2.

[56] W. D. Kelton, *Simulation with ARENA*. McGraw-hill, 2002.

[57] S. V. Rice, H. M. Markowitz, A. Marjanski, and S. M. Bailey, "The simscript III programming language for modular object-oriented simulation," in *Proceedings of the*

37th conference on Winter simulation. Winter Simulation Conference, 2005, pp. 621–630.

[58] A. A. B. Pritsker, "Introduction to stimulation and slam II," 1986.

[59] D. Krahl, "Extend: the extend simulation environment," in *Proceedings of the 34th conference on Winter simulation: exploring new frontiers*. Winter Simulation Conference, 2002, pp. 205–213.

[60] C. Udeze, K. Okafor, and C. Okezie, "Matlab simevent: A process model approach for event-based communication network design (a case for reengineered dcn)," *Journal of Basic and Applied Sciences*, vol. 2, no. 5, pp. 5070–5080, 2012.

[61] A. A. B. Pritsker, "Introduction to gasp iv," in *Proceedings of the 9th conference on Winter simulation-Volume 1*. Winter Simulation Conference, 1977, pp. 28–30.

[62] R. M. Bryant, "Simpas: A simulation language based on pascal," in *Winter Simulation Conference*, 1980.

[63] G. M. Birtwistle, *Discrete event modelling on SIMULA*. Macmillan International Higher Education, 1979.

[64] D. Bolier and A. Eliëns, "Sim : a c++ library for discrete event simulation," 12 1994.

[65] S. Cass, "The 2015 top ten programming languages," *IEEE Spectrum, July*, vol. 20, 2015.

[66] I. S. Association *et al.*, "802.11 p-2010-ieee standard for information technology-local and metropolitan area networks-specific requirements-part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6: Wireless access in vehicular environments," *URL: http://standards. ieee. org/findstds/standard/802.11 p-2010. html*, 2010.

[67] Mathworks, "802.11p packet error rate simulation for a vehicular channel," September 2018. [Online]. Available: https://www.mathworks.com/help/

wlan/examples/802-11p-packet-error-rate-simulation-for-a-vehicular-channel.html; jsessionid=5e1e82929ef44b4215b738e53a35

[68] B. Li, M. S. Mirhashemi, X. Laurent, and J. B. Gao, "Wireless access for vehicular environments," 2011.

[69] S. A. Ahmed, S. H. Ariffin, and N. Fisal, "Overview of wireless access in vehicular environment (wave) protocols and standards," *Indian Journal of Science and Technology*, vol. 6, no. 7, pp. 4994–5001, 2013.

[70] A. M.S Abdelgader and W. Lenan, "The Physical layer of the IEEE 802.11p WAVE Communication Standard: The Specification and Challenges," *World Congress on Engineering and Computer Sciences*, vol. II, pp. 22–24, 2014.

[71] Y. J. Li, "An overview of the dsrc/wave technology," in *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*. Springer, 2010, pp. 544–558.

[72] "Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," Tech. Rep., March 2012.

[73] C.-K. Park, M.-W. Ryu, and K.-H. Cho, "Survey of mac protocols for vehicular ad hoc networks," *SmartCR*, vol. 2, no. 4, pp. 286–295, 2012.

[74] *J2735 Dedicated Short Range Communications (DSRC) Message Set Dictionary*, SAE Std.

[75] *ETSI TR 102 638, Intelligent Transport System (ITS); vehicular communications; basic set of applications; definition, ETSI specification TR 102 638, Version 1.1.1.*, ETSI Std.

[76] S. Yousefi, M. Fathy, and A. Benslimane, "Performance of beacon safety message dissemination in Vehicular Ad hoc NETworks (VANETs)," *Journal of Zhejiang University SCIENCE A*, vol. 8, no. 12, pp. 1990–2004, 2007.

[77] M. Khabazian, S. Aissa, and M. Mehmet-Ali, "Performance modeling of safety messages broadcast in vehicular ad hoc networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 1, pp. 380–387, 2013.

[78] Y. Yao, L. Rao, X. Liu, and X. Zhou, "Delay analysis and study of IEEE 802.11p based DSRC safety communication in a highway environment," *Proceedings - IEEE INFOCOM*, pp. 1591–1599, 2013.

[79] *J2945 On-Board System Requirements for V2V Safety Communications*, SAE Std.

[80] *IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Multi-Channel Operation*, Std., March 2016.

[81] *IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Std., Dec 2016.

[82] "Vehicles in network simulation (VeinS)." [Online]. Available: http://veins.car2x.org

[83] "iTETRIS: The open simulation paltform for intelligent transport system (its) services." [Online]. Available: http://www.ict\protect\discretionary{\char\hyphenchar\font}{}{}itetris.eu/itetris_platform.html

[84] "VSimRTI - smart mobility simulation." [Online]. Available: https://www.dcaiti.tu\protect\discretionary{\char\hyphenchar\font}{}{}berlin.de/research/simulation/

[85] S. Papanastasiou, J. Mittag, E. G. Strom, and H. Hartenstein, "Bridging the gap between physical layer emulation and network simulation," in *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*. IEEE, 2010, pp. 1–6.

[86] B. Aygun, "Distributed adaptation techniques for connected vehicles," Ph.D. dissertation, WPI, Worcester,MA, 8 2016.

[87] M. Boban, J. Barros, and O. Tonguz, "Geometry-based vehicle-to-vehicle channel

modeling for large-scale simulation," *IEEE Transactions on Vehicular Technology*, vol. 63, no. 9, pp. 4146–4164, Nov 2014.

[88] E. Perahia and R. Stacey, *Next Generation Wireless LANs: 802.11n and 802.11ac*, 2nd ed. Cambridge University Press, 2013.

[89] E. Sourour, H. El-Ghoroury, and D. McNeill, "Frequency offset estimation and correction in the ieee 802.11a wlan," in *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*, vol. 7, Sep. 2004, pp. 4923–4927 Vol. 7.

[90] S. Biddlestone and K. A. Redmill, "A gnu radio based testbed implementation with ieee 1609 wave functionality," in *2009 IEEE Vehicular Networking Conference (VNC)*, Oct 2009, pp. 1–7.

[91] C. Michaels, "DSRC Implementatio Guide - A guide to users of SAE J2735 message sets over DSRC," SAE, Tech. Rep., Feb 2010.

[92] A. Vinel, N. Lyamin, and P. Isachenkov, "Modeling of v2v communications for c-its safety applications: A cps perspective," *IEEE Communications Letters*, vol. 22, no. 8, pp. 1600–1603, Aug 2018.

[93] T. ElBatt, S. K. Goel, G. Holland, H. Krishnan, and J. Parikh, "Cooperative collision warning using dedicated short range wireless communications," *Proceedings of the 3rd international workshop on Vehicular ad hoc networks - VANET '06*, p. 1, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1161064.1161066

[94] S. D. Patil, D. Thombare, and V. D. Khairnar, "Simulation of realistic mobility model and implementation of 802.11 p (dsrc) for vehicular networks (vanet)," *arXiv preprint arXiv:1304.5190*, 2013.

[95] P. Gipps, "A model for the structure of lane-changing decisions," *Transportation Research Part B: Methodological*, vol. 20, no. 5, pp. 403 – 414, 1986. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0191261586900123

[96] J. Nilsson, M. Brannstrom, E. Coelingh, and J. Fredriksson, "Lane Change Maneuvers for Automated Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1087–1096, 2017.

[97] M. Keyvan-Ekbatani, V. L. Knoop, and W. Daamen, "Categorization of the lane change decision process on freeways," *Transportation Research Part C: Emerging Technologies*, vol. 69, pp. 515–526, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.trc.2015.11.012

[98] Mathworks, "Matlab: The language of technical computing," https://www.mathworks.com/tagteam/73554˙91199v02˙overview.pdf, 2012.

[99] M. Lacage and T. R. Henderson, "Yet another network simulator," in *Proceeding from the 2006 workshop on ns-2: the IP network simulator*. ACM, 2006, p. 12.

[100] MathWorks, "AWGN Channel." [Online]. Available: https://www.mathworks.com/help/comm/ug/awgn\protect\discretionary{\char\hyphenchar\font}{}{}channel\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{h\global\mathchardef\accent@spacefactor\spacefactor}\accent95h\egroup\spacefactor\accent@spacefactortml

[101] J. A. Fernandez, D. D. Stancil, and F. Bai, "Dynamic channel equalization for ieee 802.11 p waveforms in the vehicle-to-vehicle channel," in *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*. IEEE, 2010, pp. 542–551.

[102] P. Alexander, D. Haley, and A. Grant, "Cooperative intelligent transport systems: 5.9-ghz field trials," *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1213–1235, 2011.

[103] J. Akhtman and L. Hanzo, "Decision directed channel estimation employing projection approximation subspace tracking," in *2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring*, April 2007, pp. 3056–3060.

[104] V. D. Khairnar and S. N. Pradhan, "Simulation based evaluation of highway road

scenario between dsrc/802.11 p mac protocol and stdma for vehicle-to-vehicle communication," *Journal of Transportation Technologies*, vol. 3, no. 01, p. 88, 2013.

[105] S. Eichler, C. Networks, and T. U. München, "Performance Evaluation of the IEEE 802 . 11p WAVE Communication Standard," *Vehicular Technology Conference*, pp. 2199–2203, 2007.

[106] Static and kinetic friction. [Online]. Available: http://ffden-2.phys.uaf.edu/211_fall2002.web.dir/ben_townsend/staticandkineticfriction.htm

[107] NHTSA, "Federal motor vehicle safety standards; stopping distance table," September 1999. [Online]. Available: https://www.gpo.gov/fdsys/pkg/FR-1999-09-07/pdf/99-23226.pdf

[108] A. Kesting, M. Treiber, and D. Helbing, "Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1928, pp. 4585–4605, 2010.

[109] B. Higgs, M. M. Abbas, and A. Medina, "Analysis of the wiedemann car following model over different speeds using naturalistic data," in *Procedia of RSS Conference*, 2011, pp. 1–22.

[110] A. Schadschneider, "The nagel-schreckenberg model revisited," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 10, no. 3, pp. 573–582, 1999.

[111] S. Krauß, "Towards a unified view of microscopic traffic flow theories," *IFAC Proceedings Volumes*, vol. 30, no. 8, pp. 901–905, 1997.

[112] IEEE, "IEEE Guide for Wireless Access in Vehicular Environments (WAVE) - Architecture," *IEEE Std 1609.0-2013*, pp. 1–78, March 2014.

[113] L. Wang, R. F. Iida, and A. M. Wyglinski, "Performance analysis of EDCA for ieee 802.11p/dsrc based v2v communication in discrete event system," in *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, Sept 2017, pp. 1–5.

[114] "IEEE approved draft standard for wireless access in vehicular environments (wave) - networking services," *IEEE P1609.3v3/D6, November 2015*, pp. 1–162, Jan 2016.

[115] I. Noblis, "Vehicle Information Exchange Needs for Mobility Applications Version 3.0 ," ITS, Tech. Rep., April 2013. [Online]. Available: http://www.its.dot.gov/index.htm

[116] M. Gramaglia, "VANET-based optimization of infotainment and traffic efficiency vehicular services," 2012.

[117] L. Wang, R. F. Iida, and A. M. Wyglinski, "Coordinated lane changing using v2v communications," Aug 2018.

[118] T. K. Mak, K. P. Laberteaux, R. Sengupta, and M. Ergen, "Multichannel medium access control for dedicated short-range communications," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 1, pp. 349–366, Jan 2009.

[119] C. Song, "Performance analysis of the IEEE 802.11 p multichannel mac protocol in vehicular ad hoc networks," *Sensors*, vol. 17, no. 12, p. 2890, 2017.

[120] Q. Chen, D. Jiang, and L. Delgrossi, "IEEE 1609.4 DSRC multi-channel operations and its implications on vehicle safety communications," in *Vehicular Networking Conference (VNC), 2009 IEEE*. IEEE, 2009, pp. 1–8.

[121] C. Campolo, A. Molinaro, A. Vinel, and Y. Zhang, "Modeling prioritized broadcasting in multichannel vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 2, pp. 687–701, 2012.

[122] J. Guo and N. Balon, "Vehicular ad hoc networks and dedicated short-range communication," *University of Michigan*, 2006.

[123] Q. Wang, S. Leng, H. Fu, and Y. Zhang, "An ieee 802.11p-based multichannel mac scheme with channel coordination for vehicular ad hoc networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 2, pp. 449–458, June 2012.

[124] E. Behravesh and A. Butler, "Evaluation of the ieee 802.11 p multi-channel operation in vehicular networks," PeerJ Preprints, Tech. Rep., 2016.

[125] C. Song, G. Tan, and C. Yu, "An efficient and QoS supported multichannel MAC protocol for vehicular ad hoc networks," *Sensors*, vol. 17, no. 10, p. 2293, 2017.

[126] C. Campolo and A. Molinaro, "Cooperative multichannel management in IEEE 802.11 p/WAVE vehicular ad hoc networks," *International Journal of Vehicle Information and Communication Systems*, vol. 2, no. 3-4, pp. 147–176, 2011.

[127] FCC, "06-110 Memorandum Opinion and Order," Tech. Rep., 2016.

[128] S. Mangold, S. Choi, P. May, O. Klein, G. Hiertz, and L. Stibor, "Ieee 802.11 e wireless lan for quality of service," in *Proc. European Wireless*, vol. 2, 2002, pp. 32–39.

[129] K. Bilstrup, E. Uhlemann, E. G. Strom, and U. Bilstrup, "Evaluation of the ieee 802.11 p mac method for vehicle-to-vehicle communication," in *2008 IEEE 68th Vehicular Technology Conference*. IEEE, 2008, pp. 1–5.

[130] A. J. Ghandour, M. Di Felice, L. Bononi, and H. Artail, "Modeling and simulation of wave 1609.4-based multi-channel vehicular ad hoc networks," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and ..., 2012, pp. 148–156.

[131] S.-e. Chung, J. Yoo, and C.-k. Kim, "A cognitive MAC for VANET based on the WAVE systems," in *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, vol. 1. IEEE, 2009, pp. 41–46.

[132] J. Chu, K. Feng, C. Chuah, and C. Liu, "Cognitive radio enabled multi-channel access for vehicular communications," in *2010 IEEE 72nd Vehicular Technology Conference - Fall*, Sep. 2010, pp. 1–5.

[133] L. De Martini and J. Härri, "Short paper: Design and evaluation of a multi-channel mechanism for vehicular service management at 5.9ghz," in *2013 IEEE Vehicular Networking Conference*, Dec 2013, pp. 178–181.

[134] Y. Zang, L. Stibor, B. Walke, H. Reumerman, and A. Barroso, "A novel mac protocol for throughput sensitive applications in vehicular environments," in *2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring*, April 2007, pp. 2580–2584.

[135] F. Klingler, F. Dressler, J. Cao, and C. Sommer, "Use both lanes: Multi-channel beaconing for message dissemination in vehicular networks," in *2013 10th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, March 2013, pp. 162–169.

[136] S.-w. Chang, J. Cha, and S.-s. Lee, "Adaptive edca mechanism for vehicular ad-hoc network," in *The International Conference on Information Network 2012.* IEEE, 2012, pp. 379–383.

[137] C. Campolo, A. Cortese, and A. Molinaro, "CRaSCH: A cooperative scheme for service channel reservation in 802.11 p/wave vehicular ad hoc networks," in *2009 International Conference on Ultra Modern Telecommunications & Workshops.* IEEE, 2009, pp. 1–8.

[138] M. R. Stiglitz and C. Blanchard, "Ieee standard dictionary of electrical and electronic terms," *Microwave Journal*, vol. 35, no. 4, pp. 150–151, 1992.